MASTER EN SCIENCES INFORMATIQUES

Cyrano
réalisation d'un synthétiseur vocal

Gailly, Geoffroy

Award date:
1997

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX
NAMUR

Institut d'informatique

CYRANO

Réalisation d'un synthétiseur vocal

-

Geoffroy Gailly

Promoteur : Professeur Claude Cherton

rue Grandgagnage, 21 ● B-5000 Namur (Belgium)

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX
NAMUR

Institut d'informatique

CYRANO

Réalisation d'un synthétiseur vocal

-

Geoffroy Gailly

Promoteur : Professeur Claude Cherton

rue Grandgagnage, 21 ● B-5000 Namur (Belgium)

Je souhaiterais remercier chaleureusement mon promoteur,
le professeur Claude Cherton,
pour l'aide et le support
qu'il m'a apportés tout au long de mon mémoire.

Je souhaiterais également remercier
monsieur Stéphane Lamy,
du centre 'La Famille' de Bruxelles,
pour son aide durant la réalisation de ce travail.

Je remercie de plus
toute l'équipe de développement du programme Mbrola
de la Faculté Polytechnique de Mons,
pour le moteur vocal de grande qualité qu'ils ont su créer.

Je remercie enfin Danièle et ma famille
pour leur soutient moral
dans les moments difficiles.

A Edmond

TABLE DES MATIERES

INTRODUCTION

Il y a cent ans, Rostand nous offrait *Cyrano*...

A l'époque, il y a peu de chances que l'on ait vu, dans le personnage de Christian, l'un des premiers exemples de synthétiseur vocal. Et pourtant, qu'a-t-il fait d'autre que de répéter tout haut, pour l'amour de Roxane, ce que Cyrano lui glissait tout bas ?

> « *Veux-tu que nous fassions - et bientôt tu l'embrases ! -*
> *Collaborer un peu tes lèvres et mes phrases ?* »
>
> E. Rostand, Cyrano de Bergerac,
> Deuxième acte - Scène X

Aujourd'hui, nous nous proposons, avec les moyens de l'informatique moderne, de remplacer l'encombrant Christian par un ordinateur.

L'idée qui nous est venue est d'utiliser les possibilités sonores des ordinateurs individuels pour leur apprendre à parler. Nous souhaiterions créer un programme capable d'exprimer verbalement ce qu'une personne lui communiquerait par écrit.

C'est avec cette idée en tête que nous avons rencontré monsieur Stéphane Lamy, du centre *La Famille* de Bruxelles. Il travaille avec des handicapés, parmi lesquels des personnes privées de la parole. Notre proposition l'a beaucoup intéressé.

A notre connaissance, dans l'état actuel des choses, il n'existe pas d'aides informatiques à la communication de qualité en français.

La mise en commun de nos idées a permis de donner naissance au travail que vous avez dans les mains : la réalisation d'un synthétiseur vocal.

Au cours des pages qui vont suivre, nous commencerons par décrire, par le biais des besoins de l'utilisateur, le programme que nous souhaitons réaliser (chapitre 1). Nous verrons ensuite à quel public nous nous adressons (chapitre 2). Un chapitre sera alors consacré à l'aspect méthodologique de notre travail (chapitre 3). Les trois chapitres suivants vous décriront l'implémentation des différents éléments du logiciel (chapitres 4, 5 et 6). Nous évaluerons alors le programme avec monsieur Lamy ainsi que les premières personnes à avoir pu l'essayer (chapitre 7). Il nous restera à envisager le futur du programme (chapitre 8). Nous conclurons ce mémoire d'une manière plus personnelle en essayant de faire le point sur l'expérience du mémoire vécue de l'intérieur.

CHAPITRE 1
LES BESOINS DE L'UTILISATEUR

Lorsque le professeur Claude Cherton et moi avons rencontré monsieur Stéphane Lamy pour la première fois, nous lui avons proposé de réaliser, avec son aide, un synthétiseur vocal informatique destiné aux personnes privées de la parole.

Nous avons donc décidé de développer un programme permettant de prononcer n'importe quelle suite de phonèmes via un haut-parleur. Le but de ce programme était de donner à l'utilisateur la capacité de s'exprimer de manière sonore aussi naturellement que possible.

Une fois ce but clairement exprimé, monsieur Lamy, le professeur Cherton et moi-même nous sommes rencontrés à deux reprises afin de mettre sur papier tous les souhaits que nous pouvions avoir concernant le logiciel à développer.

1.1 Les souhaits exprimés

De nombreuses idées ont été lancées au cours de nos réunions. Je les ai rangées en deux catégories : *les fonctionnalités du programme* et *la voix et la prononciation*.

Les fonctionnalités du programme

Dans cette catégorie sont placées les idées concernant les services de base que le programme devra être à même de fournir.

- Utiliser des phonèmes comme élément de base pour la construction de séquences sonores permettrait de donner à l'utilisateur une grande liberté d'expression. En effet, on peut percevoir intuitivement que tous les mots ou onomatopées qu'une personne peut souhaiter prononcer sont décomposables en phonèmes.

- Monsieur Lamy a proposé d'adapter, sous forme informatique, la méthode *PHONEPIC* développée par la Fondation Suisse pour les Téléthèses (F.S.T.). Grâce à cette méthode, l'apprentissage et l'utilisation de phonèmes est facilitée pour les enfants et les handicapés. Dans notre programme, nous essayerons donc d'intégrer une interface présentant le clavier à pictogrammes de la F.S.T.

- Les utilisateurs risquent d'avoir des niveaux très variables en dactylographie. Il faudrait que le logiciel offre des modes de fonctionnement différents adaptables à l'utilisateur.

Il faudrait, par exemple, que, pour une personne peu habile avec le clavier, il soit possible de composer une phrase phonétique complète pour en demander ensuite la prononciation.

Pour une personne tapant plus vite à la machine, il faudrait laisser la possibilité de prononcer chaque phonème dès que la touche du clavier correspondant à ce phonème est frappée.

- Pour accélérer le débit de la prononciation, peut-être serait-il possible que l'utilisateur prépare à l'avance un certain nombre de phrases phonétiques auxquelles il pourrait accéder simplement et rapidement (grâce à la souris par exemple) en cours d'utilisation du logiciel. Bien entendu, cette fonctionnalité serait proposée en supplément de la possibilité initiale de taper une suite de phonèmes à prononcer immédiatement.

  Cette idée a vu le jour pour deux raisons. Tout d'abord, un certain nombre de mots et de phrases reviennent régulièrement dans le discours quotidien d'une personne. La préparation de ces mots et de ces phrases diminuerait le temps passé à taper les phonèmes en cours de conversation. La deuxième raison est que l'étape de la dactylographie des mots et phrases à prononcer peut prendre beaucoup de temps si l'utilisateur n'a pas l'habitude du clavier. Lui permettre de composer ses phrases à son aise pour pouvoir les prononcer plus tard pourrait alors être la seule manière pour lui d'exploiter le synthétiseur de manière satisfaisante.

- Cette notion de préfabrication (point précédent) pourrait même être étendue pour permettre à l'utilisateur de ranger ses phrases préfabriquées dans des groupes ou contextes particuliers. L'utilisateur pourrait alors à chaque instant demander à accéder au contenu d'un groupe donné pour pouvoir prononcer une ou plusieurs des phrases qu'il contient.

  Grâce à cette fonctionnalité, l'utilisateur aurait plus de facilité à gérer ses phrases préfabriquées. Si le nombre de celles-ci devient trop important, le fait de pouvoir les regrouper par thème devrait permettre de les retrouver plus simplement et rapidement car l'effort de mémoire nécessaire pour retrouver la bonne phrase au moment opportun serait réduit.

- Toujours dans le même ordre d'idées, nous pourrions donner à l'utilisateur la possibilité d'établir entre les groupes créés des relations permettant de voyager d'un groupe à l'autre. Il pourrait dès lors établir une véritable structure de groupes interreliés dans laquelle il pourrait se déplacer à son gré, prononçant les phrases qu'il souhaite au moment où il le souhaite.

- A chacun de ces groupes et phrases pourraient être associés des raccourcis, suites de caractères permettant d'augmenter quelque peu la vitesse de déplacement dans la structure et la vitesse d'accès aux phrases préfabriquées à prononcer.

- Une liste des phrases phonétiques prononcées pendant l'utilisation courante du logiciel devrait être disponible et éventuellement pouvoir être imprimée. Grâce à cette liste, l'utilisateur pourrait se rendre compte des erreurs qu'il commet lorsqu'il tape les phonèmes à prononcer.

  Cette fonctionnalité pourrait être utile aussi bien pour un thérapeute travaillant avec un handicapé et souhaitant lui indiquer les erreurs commises que pour un utilisateur indépendant se rendant compte d'un défaut dans la prononciation.

- Le programme devrait être supporté par une aide contextuelle complète et compréhensible.

- Si nous choisissions d'utiliser des pictogrammes pour représenter les phonèmes à l'écran, nous pourrions également donner à l'utilisateur un outil permettant de modifier ces pictogrammes à volonté.

  Cette fonctionnalité permettrait à l'utilisateur de personnaliser son synthétiseur et de choisir, pour chaque phonème, un pictogramme qu'il considérerait comme plus représentatif du son associé.

La voix et la prononciation

Dans cette catégorie sont placées les idées concernant plus précisément la voix, son type et sa forme. On y retrouve donc des considérations d'ordre qualitatif sur la production sonore du logiciel.

- Il serait utile que l'utilisateur puisse choisir parmi différents types de voix: masculine ou féminine, enfantine ou adulte. De cette manière, l'utilisateur aurait plus de facilités à percevoir le logiciel comme un prolongement de lui-même et pourrait mieux accepter ce nouvel organe.

  De la même manière, la langue d'expression pourrait être choisie. Les phonèmes varient en effet d'une langue à l'autre.

- Le ton de voix traduit autant de choses que les phonèmes prononcés. Il faudrait dès lors laisser la possibilité de prononcer les phrases avec un ton de voix plus enjoué ou plus lugubre, plus triste ou plus gai.

- Afin de permettre des formulations de phrases très diverses telles que des secrets chuchotés ou des injures hurlées, le volume de la prononciation devrait être réglable rapidement et simplement. Bien que bon nombre d'enceintes pour ordinateur offrent déjà cette fonctionnalité, le réglage logiciel du volume pourrait constituer un plus.

1.2 Les contraintes du logiciel

Au cours de ces rencontres préalables au développement du logiciel, de nombreux éléments ont été mis en évidence. Deux contraintes notamment sont apparues dans l'ensemble des discussions. Elles concernent les performances du logiciel et le type très particulier de public que cible le programme.

Les performances

Lorsqu'une personne n'est pas en mesure d'exprimer une idée oralement, il va en général lui falloir plus longtemps pour exprimer son idée à l'aide d'un autre mode de communication. Or les êtres humains ne sont pas habitués à être en face d'une personne ayant besoin d'un certain laps de temps lors de la formulation de sa pensée.

Bien souvent, lorsque nous nous trouvons face à quelqu'un qui a des difficultés pour parler (un bègue par exemple), nous ne prenons pas le temps de laisser la personne aller jusqu'au bout de sa tentative de s'exprimer. Le réflexe est de compléter les paroles non terminées dès que la signification est (croit-on) perçue. Or, souvent, ce que nous prenons pour une bonne action est très décourageant pour celui ou celle qui fait tant d'efforts pour surmonter son handicap.

Face à un handicapé du langage, certains s'irritent de la lenteur d'expression, d'autres cherchent à éviter la conversation, d'autres encore font tout leur possible pour décharger le handicapé du besoin de s'exprimer longuement, un très petit nombre de personnes prend le temps de laisser le handicapé formuler sa pensée.

Si la prothèse vocale que nous cherchons à fabriquer ne permet pas à l'utilisateur de s'exprimer de manière relativement fluide et rapide, le programme sera vite abandonné au profit d'un autre mode de communication, peut-être de moindre qualité sonore - voire même non sonore comme la langue des signes - mais offrant plus de fluidité d'expression.

Un facteur crucial de la qualité du programme final sera donc ses performances en terme de rapidité et de fluidité d'expression.

Les utilisateurs

Nous aurons l'occasion de revenir plus loin sur les publics très divers et particuliers du logiciel. C'est justement cette diversité d'utilisateurs qui pose problème ici. Nous savons en effet qu'une interface est en général adaptée à un type de public précis. Nous savons également qu'une interface adaptée à un public peut ne pas l'être du tout pour un autre public.

La difficulté lors de la conception de l'interface de notre programme va donc être de parvenir à satisfaire les utilisateurs, sachant que les besoins varient d'un utilisa-

teur à l'autre. L'ergonomie du logiciel devra être adaptée à un maximum de publics.

Nous consacrerons le chapitre 2 à la résolution des problèmes que peut poser la variété d'utilisateurs potentiels de notre programme.

Une autre caractéristique majeure des utilisateurs de notre programme est que nous ne pouvons faire aucune hypothèse sur leurs connaissances en informatique. Certains utilisateurs n'auront peut-être jamais utilisé d'ordinateur lorsque nous leur présenterons notre programme.

Nous devrons donc concevoir le logiciel et son interface en pensant à faciliter la prise en main, même par un néophyte de l'informatique. L'aide offerte par le programme devra tenir compte aussi bien des débutants que des utilisateurs aguerris.

La gestion des contraintes

En ce qui concerne la partie plus technique de la recherche de performances, seul le programmeur (ou le concepteur) est en mesure d'influencer la rapidité et la fluidité de la production vocale. Il sera donc important, lors du développement du logiciel, de porter son attention sur la conception d'un moteur vocal performant.

En ce qui concerne la maîtrise du programme, deux choses vont influencer la capacité de l'utilisateur à se servir du programme de manière satisfaisante. D'une part l'ergonomie du logiciel et, d'autre part, l'utilisateur lui-même.

D'une manière générale, l'ergonomie du logiciel facilitera la prise en main du programme. Si cette ergonomie est bien étudiée, elle permettra une meilleure connaissance et maîtrise des fonctionnalités du logiciel, qui seront alors accessibles plus intuitivement. Grâce à elle, le travail intellectuel nécessaire à l'utilisateur pour concevoir une phrase phonétique sera réduit. Il en résultera donc une plus grande rapidité et fluidité de production vocale.

D'autre part, pour un programme aussi particulier qu'une prothèse vocale, on peut s'attendre, si l'utilisateur est satisfait par le logiciel, à ce qu'il l'utilise très régulièrement. Cette fréquence d'utilisation va naturellement entraîner une bonne connaissance des capacités du programme. Si la maîtrise qu'a l'utilisateur du programme s'accroît, le débit de production vocale augmentera automatiquement. Cette augmentation sera également accrue par la familiarité avec l'outil d'interaction (clavier ou souris).

Nous devrons donc faciliter la maîtrise du programme en développant une interface de qualité, facilitant l'apprentissage des fonctionnalités offertes.

1.3 Le programme à naître

Tout au long de ce chapitre, nous avons essayé de dessiner le cadre du programme que nous voulons développer. Dans les pages qui vont suivre, nous allons tenter de réaliser, à travers le logiciel, un maximum des souhaits formulés au point 1.1.

Nous viserons, à travers notre réalisation, à privilégier la rapidité et la fluidité de la prononciation. Bien entendu, nous ne négligerons pas la qualité de l'interface, qui devra tenir compte du public visé. Le programme devra, en outre, permettre l'apprentissage complet de ses fonctionnalités par l'utilisateur grâce à une aide contextuelle complète.

CHAPITRE 2
LES PUBLICS DU LOGICIEL

2.1 Les handicapés du langage

Le logiciel que nous développons s'adresse aux handicapés du langage. C'est à dire, en d'autres mots, à toute personne qui, pour une raison ou pour une autre, n'est pas capable de s'exprimer oralement naturellement dans sa langue maternelle.

Cette définition des utilisateurs potentiels du programme regroupe un grand nombre de personnes. Parmi ces personnes, on trouve aussi bien des muets de naissance que des handicapés moteurs incapables de formuler verbalement leurs pensées. On trouve également aussi bien des adultes que des enfants. Comme on le voit, les publics sont nombreux et variés.

Les publics sont tellement variés qu'il serait impossible de répertorier tous leurs besoins, tant il y en aurait à prendre en compte. Certains utilisateurs auraient besoin d'un programme très simple, d'autres souhaiteraient un accompagnement plus poussé au niveau de l'interface. Pour une partie, un champ d'édition et un bouton de prononciation suffiraient, pour d'autres, un système de pictogrammes serait la seule manière de comprendre et de maîtriser la notion de phonèmes. Certains utilisateurs potentiels n'ont pas l'usage de leurs membres; un système de balayage d'une représentation du clavier à l'écran serait pour eux la seule manière de pouvoir composer les phrases qu'ils souhaitent prononcer.

Pour bien faire, il faudrait considérer séparément le cas de chacune de ces personnes privées de la parole pour pouvoir établir quels sont précisément ses besoins et réaliser pour elle un programme sur mesure.

Si nous nous lançons dans la réalisation d'un programme permettant de satisfaire tous les utilisateurs potentiels, il y a fort à parier que le résultat sera décevant. Un logiciel permettant de répondre à tant de souhaits simultanément serait, à n'en pas douter, extrêmement complexe, au moins au niveau de la paramétrisation du système.

Cette complexité est-elle souhaitable ? Probablement pas. En effet, son seul but est d'offrir un maximum d'adaptabilité de l'interface et des fonctionnalités du programmes. Or, une fois la paramétrisation du système effectuée, toute la complexité du programme devient un fardeau inutile. Tout au plus, l'utilisateur pourra vouloir modifier l'interface ou ses fonctionnalités une ou deux fois au cours des utilisations futures.

Il serait bien plus intéressant de développer des programmes séparés, résolvant chacun un type de problème rencontré. Un programme pourrait ainsi être destiné aux personnes muettes mais n'ayant aucun autre handicap. Un autre pourrait être

orienté vers la résolution des problèmes rencontrés par les handicapés physiques ou mentaux lorsqu'ils souhaitent s'exprimer. Un autre encore pourrait s'adresser aux personnes nécessitant une aide active pour la composition des phrases phonétiques à prononcer (systèmes à balayage ou autre). On pourrait alors développer toute une panoplie de produits différents s'adressant chacun à un public différent. Chaque produit (programme) serait alors suffisamment ciblé pour n'être pas trop complexe.

Dans le cadre qui nous occupe, choisir un public et développer le programme y correspondant sera sans doute tout ce que le temps nous autorisera à faire. Voyons donc quel public nous allons choisir et, à partir de ce choix, quel logiciel nous allons réaliser pour répondre aux problèmes posés.

2.2 Le choix d'un public

Il nous faut choisir un public parmi tous ceux que concerne le problème du mutisme. La première question qui vient alors à l'esprit est : "Avons nous plus de raison de choisir un public plutôt qu'un autre ?". Pour répondre à cette question, il nous faut tout d'abord considérer les travaux ayant déjà été réalisés dans le domaine, afin de ne pas refaire ce qui existe et de pouvoir répondre aux besoins restés sans réponse.

Or, justement, les synthétiseurs de parole ne courent pas les rues. Il n'existe, pour l'instant, que très peu de logiciels pouvant nous servir de base de travail. De nombreux publics sont encore à satisfaire.

Toutefois, certains publics sont plus exigeants que d'autres. La satisfaction de certains d'entre eux demande un travail de recherche et de développement considérable en plus de la charge concernant la réalisation d'un moteur vocal. Si nous souhaitons pouvoir mener à bien notre entreprise, peut-être serait-il plus profitable de commencer par la recherche d'un moteur vocal de bonne qualité, assorti d'une interface destinée à un public moins exigeant.

Quel est le dénominateur commun de nos utilisateurs potentiels ? Ils sont muets. Certains ont d'autres handicaps, d'autres n'ont que celui-là. Il paraît intuitivement plus raisonnable, dans les circonstances qui nous occupent, de tenter de satisfaire, pour commencer, les utilisateurs posant le moins de problèmes.

Notre choix va donc naturellement se poser sur le public des personnes muettes n'ayant pas d'autre handicap.

2.3 Les conséquences du choix

D'une manière générale, poser ce choix nous permet d'augmenter notre capacité à atteindre l'objectif fixé, aussi bien au niveau de la panoplie de programmes dont

nous avons parlé plus haut (qui, nous l'espérons, sera un jour complète) qu'au niveau du programme précis que nous proposons de réaliser dans le cadre de ce mémoire.

Bien entendu, dans un premier temps, le public touché par le logiciel sera réduit. Toutefois, ce programme pourra servir de base pour les développements futurs axés vers des types de publics différents. Au moment de la réalisation de ces nouveaux programmes, un moteur vocal aura été trouvé et sera disponible pour les développements suivants. La charge de travail du concepteur en sera d'autant allégée.

CHAPITRE 3
LA METHODOLOGIE DU DEVELOPPEMENT

Ce chapitre, qui servira, en quelque sorte, de charnière entre l'expression des besoins et les phases plus techniques du développement. Nous commencerons par nous poser la question de l'intérêt de l'informatique face au problème qui nous est posé. Ensuite, nous verrons de quelle manière nous pouvons aborder le développement.

3.1 Le choix de l'informatique

Arrêtons-nous un instant ici pour considérer cette question extrêmement importante dans le contexte de ce mémoire: "*Le choix de l'informatique est-il un bon choix pour résoudre le problème qui nous est posé ?*". Car, en fait, si nous devions répondre par la négative, c'est l'ensemble du travail dans lequel nous nous sommes lancés qui deviendrait inutile...

Tout d'abord, l'ordinateur permet une grande interactivité entre l'utilisateur et le programme utilisé. Il pourrait dès lors être très simple de demander à la personne ce qu'elle souhaite exprimer, pour l'exprimer ensuite à sa place.

De plus, la production de sons du langage humain est possible avec un ordinateur. Les programmes de *text-to-speech* que l'on trouve actuellement sur le marché le prouvent. La production vocale est même simplifiée par les développements récents du multimédia. A l'heure actuelle, utiliser une carte son pour jouer des fichiers sonores ne pose aucun problème.

La transition entre les commandes de l'utilisateur et les sons du langage humain (mots, phrases...) est possible de manière logicielle. A nouveau, prenons ici l'exemple des programmes de *text-to-speech* qui prouvent que cette transition est possible.

On peut toutefois se demander si des solutions électroniques implémentant le programme dont nous parlons ne pourraient pas être plus intéressantes. On pourrait imaginer par exemple un petit boîtier offrant un clavier et un petit écran et permettant de prononcer des mots de la même façon que notre programme. Toutefois, il y a fort à parier que le prix de revient d'un tel équipement serait fort élevé (production à échelle réduite, coût des composants...).

De leur côté, les ordinateurs, malgré un prix également élevé, offrent une polyvalence que n'offriraient pas les solutions électroniques sur mesure. Avec les développements récents des ordinateurs personnels portables, un programme informatique serait potentiellement plus pratique pour un utilisateur qu'un boîtier électronique destiné uniquement à synthétiser la voix humaine.

Il semble donc que le choix de l'informatique soit le bon dans le cas qui nous concerne.

3.2 Le choix de la méthodologie

Au départ, lorsqu'il a été question de commencer à développer le programme, nous nous sommes orientés vers la méthodologie TRIDENT. Etant orientée tâche, cette méthodologie nous aurait permis de porter notre attention sur l'utilisateur et la tâche que nous souhaitons l'aider à réaliser plutôt que sur des traitements ou des données. Le contexte particulier du programme semblait d'ailleurs bien se prêter à un développement orienté tâche.

Nous avons donc essayé de développer l'interface de notre programme en suivant la méthodologie TRIDENT. Toutefois, nous nous sommes heurtés à un certain nombre de difficultés qui nous ont amenés à changer quelque peu notre point de vue sur le développement du programme.

Tout d'abord, il nous a été difficile de suivre pas à pas la méthodologie. Certaines étapes de développement, pourtant très importantes, ne trouvaient pas de réponse. La description de l'utilisateur, par exemple, était si vaste dans le projet Cyrano qu'il n'était simplement pas possible de la décomposer en classes. Or, on l'admettra aisément, cette décomposition est cruciale dans la méthodologie TRIDENT puisque c'est (en partie) grâce à elle que le choix d'un type d'interaction peut être fait pour le programme.

Nous sommes malgré tout parvenus au terme du développement de l'interface. Celle-ci se révéla vite très décevante. Un grand nombre d'éléments, qu'il semblait, intuitivement, important de retrouver dans nos fenêtre, avaient, tout simplement, échappé à l'analyse. Par exemple, on ne retrouvait à aucun endroit des boutons de raccourci permettant d'accéder rapidement aux fonctionnalités des menus. De la même manière, aucun objet d'aide (ligne contenant un texte d'aide par exemple) n'avaient été prévus, alors qu'il était clair, dès le début, que l'accompagnement de l'utilisateur devait être l'une des premières préoccupations lors de la conception des fenêtres. Tous ces éléments faisant défaut, il nous aurait fallut, pour chaque fenêtre, analyser les manques et y remédier manuellement.

Pour finir, il nous sembla que la charge de travail imposée par la méthodologie TRIDENT, avec, pour seul but, une quinzaine de fenêtres, sommes toutes fort simples, était beaucoup trop lourde.

C'est pourquoi il ne nous paraît pas utile de présenter ici les résultats de notre analyse. Le développement propre aux fenêtres nécessaires dans le programme sera effectué au cours de la réalisation du programme, lorsque le besoin de l'une ou l'autre de ces fenêtres se fera sentir.

3.3 Le développement

Le développement proprement dit du programme se fera dans l'ordre suivant :

- Tout d'abord, nous nous attacherons, dans le chapitre 4, à chercher, sur le marché, un moteur vocal déjà existant, capable d'effectuer la synthèse vocale sous une forme ou une autre, et que nous pourrions utiliser dans notre programme. Nous avons dit l'importance de ce moteur dans le chapitre 1.

- Nous constituerons ensuite, dans le chapitre 5, les structures de données propres à notre programme. Deux structures en particulier retiendront notre attention. D'une part nous trouverons la structure de groupes et de phrases, que nous devrons pouvoir conserver et qui devra proposer à l'utilisateur un ensemble de fonctionnalités de gestion de cette structure. D'autre part nous trouverons les paramètres utilisateurs, dont nous devrons également nous assurer de la conservation d'une utilisation du logiciel à l'autre.

- Pour finir, nous réaliserons, dans le chapitre 6, le programme principal, avec son ou ses interfaces de communication avec l'utilisateur, qui devra aussi bien offrir toutes les fonctionnalités attendue du logiciel qu'assurer le lien avec les trois modules dont nous avons parlé dans les points précédents.

- Lorsque tout cela sera terminé, nous pourrons passer à une étape d'évaluation du programme qui nous permettra, avec l'aide de monsieur Lamy et des premières personnes à avoir essayé notre programme, d'évaluer le résultat de notre travail.

CHAPITRE 4
LA RECHERCHE D'UN MOTEUR VOCAL

Dans notre programme, l'élément principal sera le moteur vocal. C'est lui qui transformera les caractères introduits par l'utilisateur en suites de sons joués par la carte son. Si le moteur vocal est de mauvaise qualité ou trop lent, c'est le programme tout entier qui en souffrira. Nous devons donc accorder toute notre attention à la recherche de ce qui est réellement le coeur de notre programme.

Nous débuterons ce chapitre par une vue d'ensemble de ce que propose l'informatique en terme de production vocale à l'heure actuelle. Nous tenterons alors de faire un choix parmi les possibilités qui nous sont offertes et nous essayerons notamment de trouver sur le marché un produit qui réponde à nos attentes. Nous en viendrons ensuite à l'implémentation du moteur.

4.1 Principe de la production vocale

4.1.1 Types de moteurs existants

A l'heure actuelle, trois grand types de production vocale existent en informatique. Toutes trois ont des particularités qui les rendent intéressantes dans certaines circonstances. Nous allons voir ici quelles sont les caractéristiques de chaque type de moteur et les services qu'il peut rendre. Nous serons ensuite à même de choisir le moteur le plus approprié pour l'utilité que nous en avons.

Le moteur phonems-to-speech

Le plus simple des moteurs existants est le *phonems-to-speech*. Comme tous les autres moteurs que nous verrons ici, son but est de créer un fichier sonore et de le jouer par le biais d'une carte son (fichier au format *.wav* par exemple). Pour ce faire, ce type de moteur utilise la notation la plus proche possible du monde sonore : les phonèmes.

Comme chacun le sait, le phonème est l'élément de base du langage. Tous les mots et les sons produits dans le discours d'une personne sont décomposables en phonèmes.

Les moteurs *phonems-to-speech* possèdent, sous une forme ou une autre, une base de données de sons. Chaque phonème est associé à une représentation visuelle (caractère ou suite de caractères). Le moteur reçoit en entrée une suite de caractères et construit un fichier sonore en juxtaposant les sons correspondant à ces caractères.

Comme on s'en rend compte immédiatement, l'intérêt premier de ce type de moteur est de laisser une grande liberté dans l'assemblage des phonèmes. Il n'y a pas de suite interdite ou impossible, la faculté d'expression est donc pratiquement illimitée.

Le moteur Text-to-speech

C'est le type de moteur le mieux connu et le plus répendu sous forme logicielle à l'heure actuelle. La raison en est simple : là où il est nécessaire d'apprendre comment communiquer avec le moteur *phonems-to-speech* (comment écrire les phonèmes avec le clavier), le moteur *text-to-speech* fait tout l'effort à votre place.

Un moteur de ce type est en effet capable de décider, dans un texte rédigé intégralement dans une langue, quelle est la prononciation de chaque mot. En appliquant un ensemble de règles à la suite de caractères du texte, le moteur est capable de savoir si une lettre dans un mot se prononce ou non, comment elle se prononce; lorsqu'elle est en fin de mot, si elle se prononce et si elle donne lieu à une liaison (et laquelle)... Bref, il s'agit d'un lecteur de texte transformant les caractères reçus en une suite de sons prononcés par la carte sonore.

Un moteur *text-to-speech* peut être basé sur un moteur *phonems-to-speech*. Il s'agit alors de transformer le texte en une suite de phonèmes, toujours en se basant sur un ensemble de règles. C'est ensuite au moteur *phonems-to-speech* de jouer son rôle comme cela a été décrit plus haut.

Ce type de produit est très utilisé pour aider les handicapés. Les aveugles notamment car il permet d'entendre l'entièreté d'un texte sous forme électronique sans qu'aucune modification du texte ne soit nécessaire[1].

Le moteur Words-to-speech

Un moteur *words-to-speech* interprète la suite de caractères qu'il reçoit comme une suite de mots séparés par des blancs. Pour chacun de ces mots, il va alors rechercher, dans un dictionnaire interne, sa prononciation exacte (par exemple exprimée par une suite de phonèmes). Il utilise alors un moteur *phonems-to-speech* pour transformer la représentation interne en suite de sons audibles.

Pour une plus grande efficacité, les mots peuvent même être préenregistrés dans des fichiers sonores. Il n'y a plus alors qu'à prononcer directement le fichier correspondant à un mot lorsque ce mot est rencontré dans un texte. Bien entendu, cette amélioration nécessite que l'on dispose de beaucoup de place mémoire ou que le vocabulaire sur lequel on travaille soit très restreint.

[1] Il n'est pas nécessaire, comme on aurait pu le croire, d'ajouter au texte des symboles aidant le moteur à savoir quelle est la prononciation des mots du texte; ce sont les règles internes d'analyses qui permettent au moteur de prendre les décisions de prononciation.

4.1.2 Le passage au format auditif

Les sons utilisés pour composer des phrases sonores peuvent se présenter sous différentes formes. Voyons un peu quelles peuvent être ces formes.

Les phonèmes

La forme la plus évidente est la forme phonème. Un son est un phonème et est conservé comme tel dans une base de données de sons. Nous sommes donc en présence d'une base de données de petite taille dans laquelle sont stockés les sons correspondants à la trentaine de phonèmes de la langue française[2]. Pour composer un mot, il n'y a qu'à juxtaposer dans un fichier sonore les phonèmes qui le composent.

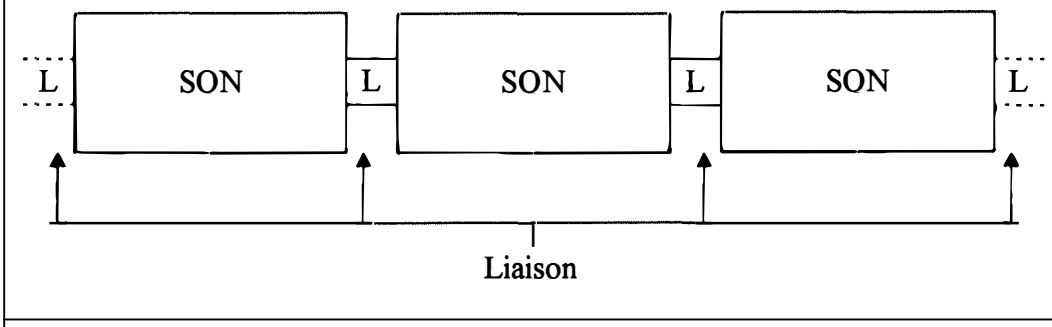

Figure 4.1 : Phrase prononcée par un être humain

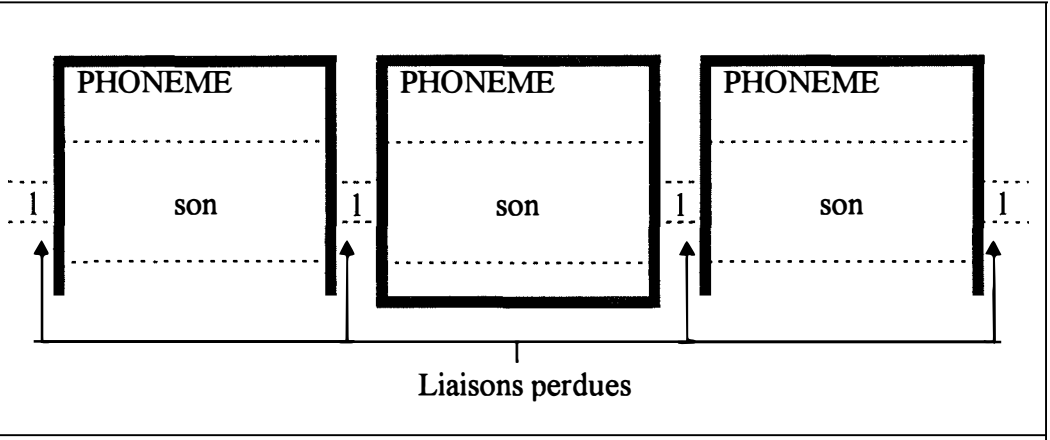

Figure 4.2 : Phrase telle que rendue par la technique des phonèmes

Toutefois, lorsqu'on écoute les résultats que donne cette méthode, on est rapidement déçu. Dans la langue française comme dans toutes les autres langues, la liaison entre deux sons est aussi importante que les sons eux-mêmes. Or cette liaison

[2] Selon l'utilisation à laquelle on destine les phonèmes, on en comptera entre 27 et 37 différents offerts dans les bases de données de la langue française.

change pour chaque paire de phonèmes possible (figure 4.1). Au moment de la prononciation des sons enregistrés, on a l'impression qu'il manque quelque chose, que les phonèmes ne sont pas bien prononcés. Lorsqu'on écoute attentivement, on se rend compte que c'est l'absence de liaison entre les phonèmes qui est gênante (figure 4.2).

Pour pallier à ce défaut, on a commencé à créer des bases de données de *diphones*. Cette technique est expliquée au point suivant.

Les diphones

Le principe des *diphones* est le même celui des phonèmes. Dans une base de données sont stockés un certain nombre d'éléments servant à la composition de phrases sonores. Toutefois, dans ce cas, chaque élément de la base de données n'est pas un phonème mais bien deux phonèmes liés par leur liaison naturelle.

Pour être précis, un *diphone* est composé de trois éléments: la seconde moitié du premier son composant le *diphone*, la liaison naturelle entre les deux sons du *diphone* et la première moitié du second son composant le *diphone*. Si les phonèmes ne sont représentés qu'à moitié, c'est pour ne pas être dupliqués au moment de la création du fichier sonore résultat. Les deux moitié du phonème constituant la fin du premier *diphone* et le début du second viennent s'ajuster parfaitement dans le fichier sonore (figure 4.3.).

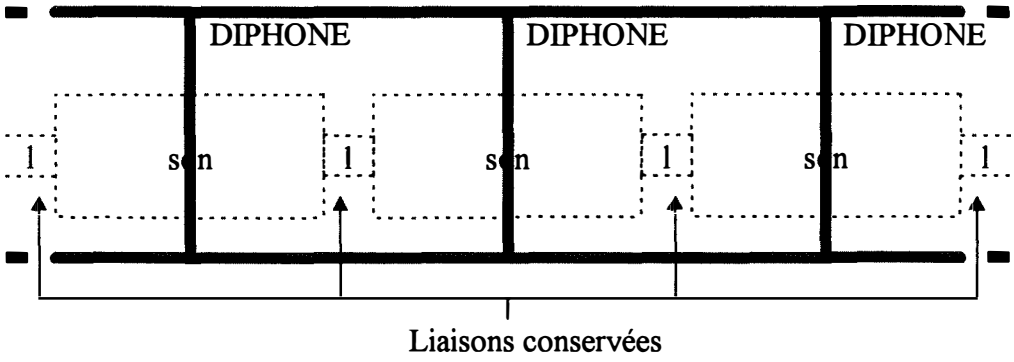

Figure 4.3 : Phrase telle que rendue par la technique des *diphones*

Le revers de la médaille est évident. En utilisant des phonèmes, nous avions besoin d'une trentaine d'enregistrements dans une base de données; en utilisant des *diphones*, nous nous retrouvons avec à peu près neuf cents (trente fois trente) éléments dans notre base de données. Une telle base de données, selon les qualités des enregistrements sonores qu'elle contient peut atteindre des tailles physiques d'environ cinq Mega-octets[3].

[3] Comme c'est la cas de la base de données française du programme *MBROLA* (voir plus loin)

Les triphones et les autres

Il n'est, bien entendu, pas obligatoire de s'arrêter à l'étape des *diphones*. Ne reculant pas devant les bases de données de taille extravagante, certains chercheurs ont tenté de créer des bases de données de *triphones* ou de *quadriphones*.

Malgré les résultats plus que satisfaisants au niveau sonore qu'offrent de telles méthodes, leurs besoins en place mémoire sont énormes: pour les *triphones*, on compte plus de vingt sept mille éléments enregistrés, soit plus de cent cinquante Mega-octets uniquement pour la base de données. On imagine le temps pris par les programmes devant accéder à ces bases de données pour composer les fichiers sonores...

4.2 Choix d'une méthode de production vocale

4.2.1 Types de moteurs adéquat

Depuis le départ, il a toujours été clair pour monsieur Lamy et pour nous que le programme devait permettre à l'utilisateur de taper les phrases à prononcer sous forme phonétique. Le moteur *phonems-to-speech* paraît donc tout désigné pour jouer le rôle de moteur vocal dans notre programme. En ce qui concerne les autres moteurs, quels pourraient être leurs avantages ?

Le moteur *text-to-speech* offre la possibilité de passer outre le besoin d'apprentissage du mécanisme de communication (les caractères phonétiques ou autres). Toutefois, il nécessite que le texte à prononcer soit écrit intégralement et, dans la mesure du possible, sans faute. La frappe est donc ralentie puisque de nombreux caractères silencieux doivent être placés dans les mots à prononcer (les indicatifs du pluriel entre autre).

De plus, la difficulté est accrue au niveau des mots n'existant pas. Pour prononcer un mot ne se trouvant pas dans le dictionnaire, il faut trouver comment l'écrire dans la langue concernée pour que le moteur, en appliquant ses règles internes, en déduise que le mot se prononce comme vous le souhaitiez.

Pour finir, le temps nécessaire à un tel moteur pour transformer une chaîne de caractères en un fichier sonore est plus important que dans un moteur *phonems-to-speech* puisqu'il y a une étape supplémentaire dans le processus (l'application des règles internes de transformation d'un texte en suite de phonèmes).

Dans le cadre qui nous occupe, le moteur *words-to-speech* est encore plus restrictif que le moteur *text-to-speech*. En effet, les seuls mots qu'il est capable de prononcer sont les mots du dictionnaire. Hors de cela, le moteur n'a aucun moyen de savoir comment un mot se prononce.

Malgré les difficultés liées à l'apprentissage des phonèmes utilisés par les moteurs *phonems-to-speech*, ce type de moteur semble clairement être le seul à répondre

aux besoins exprimés dans le premier chapitre de ce mémoire. C'est donc un moteur de ce type que nous allons maintenant tenter de trouver sur le marché des produits disponibles.

4.2.2 Le choix d'un produit sur le marché

Sur le marché des produits informatiques, il n'existe, à l'heure actuelle, pas beaucoup de développements orientés vers la production sonore. On ne trouve que quelques firmes qui se spécialisent dans le domaine. *Creatives Labs*, qui développe la carte *Sound Blaster* ainsi que *Lernout & Hauspie*, qui se spécialise également dans les cartes sonores, en sont deux exemples parmi les plus connus. A côté de cela, on trouve un ensemble de petits développements personnels plus ou moins bien réussis et plus ou moins utilisables.

On trouve aussi bien des programmes complets que des librairies d'objets ou de fonctions, voire même des *units* dans un langage ou l'autre. Faisons un peu le tri de ces programmes somme toute fort disparates. Pour commencer, lorsqu'on observe les candidats, on se rend compte que la plupart des produits sont exclusivement en anglais. Or les sons diffèrent fort entre le français et l'anglais. Conservons donc uniquement les produits disposant d'un module français ou travaillant directement dans cette langue. Déjà, la liste des produits se rétrécit fortement.

Dans notre cas, nous avons besoin d'un logiciel qui puisse s'intégrer complètement dans notre programme. Sont donc à proscrire tous les programmes ne fonctionnant que grâce à un dialogue interne permettant de choisir la suite de caractères à traiter. A nouveau, notre liste se réduit considérablement.

Comme nous l'avons vu au point 4.2.1., le type de moteur est fort important. Cependant, il faut conserver à l'esprit qu'un moteur *text-to-speech* de qualité pourrait nous satisfaire bien mieux qu'un moteur *phonems-to-speech* médiocre. Ne rayons donc pas trop vite de la liste les moteurs de ce type.

A ce point, peu de choix nous est encore proposé. Nous avons retenu deux programmes pouvant faire l'affaire parmi ceux que nous avions trouvé au départ.

Le premier est le logiciel *text-to-speech* de *Lernout & Hauspie* nommé sobrement *TTS*. La langue d'expression peut être choisie grâce à différents modules disponibles. La version que nous avons essayée était une version de démonstration mais les résultats étaient de bonne qualité.

Le second programme est développé par la Faculté Polytechnique de Mons. Il s'agit de *MBROLA*, un programme *phonems-to-speech* développé originellement par Thierry Dutoit et destiné aux applications telles que celle que nous réalisons ici. La qualité sonore du résultat (*MBROLA* travaille avec des *diphones*) est excellente.

Le fonctionnement de *MBROLA* est un peu moins simple que celui de *TTS*. Il faut commencer par composer un fichier texte dans lequel sont placés les phonèmes à prononcer, avec leur durée. *MBROLA* prend alors le fichier et le transforme en un

fichier *.wav*. A l'utilisateur alors de décider ce qu'il souhaite faire avec le fichier sonore (qu'il n'a toujours pas entendu). Pour lancer le programme, on utilise une commande *DOS* composée du nom du programme, du nom de la base de données sonores à utiliser, du nom du fichier dans lequel se trouve la suite de phonèmes à prononcer et du nom du fichier sonore résultat attendu[4].

Les deux solutions sont attrayantes. *TTS* est simple d'emploi et fourni des fichiers sonores de bonne qualité. Malheureusement il ne travaille qu'avec du texte (pas de phonèmes). *MBROLA* quant à lui est plus difficile d'emploi (création d'un fichier, choix de la durée des phonèmes et de la hauteur de ton puis lancement du programme) mais répond mieux à nos attentes et produit une synthèse de bonne qualité également.

Le critère décisif pourrait alors être financier. *TTS* est un produit commercial. Il n'est possible de l'exploiter qu'en en ayant payé le droit (achat de licence). *MBROLA* quant à lui est entièrement gratuit. Il est, de plus, développé en Belgique par des chercheurs dont le but est proche du notre (répondre aux besoins des handicapés).

Il nous semble donc raisonnable de donner la préférence à *MBROLA*. Un argument supplémentaire est sa polyvalence. Le programme peut, en effet, travailler dans n'importe quelle langue. Il suffit de lui indiquer dans quelle base de données[5] de *diphones*[6] il doit aller chercher les sons à utiliser.

4.3 Implémentation du moteur vocal

Maintenant que nous avons choisi avec quel outil travailler, nous pouvons commencer à implémenter la *unit* qui nous servira de moteur vocal dans notre programme. Elle jouera le rôle d'interface entre les caractères saisis par l'utilisateur et le programme *MBROLA* afin de créer, sur base de ce que l'utilisateur aura tapé au clavier, le fichier sonore à prononcer. Cette *Unit* offrira également une fonctionnalité permettant de demander la prononciation d'un fichier sonore donné.

4.3.1 Fonctionnalités attendues

On peut attendre trois fonctionnalités de la part de cette *unit*. Les deux premières concerneront la création des fichiers sonores (interface avec le programme *MBROLA*) et la prononciation de ces fichiers. La troisième, moins importante, permettra de comuniquer à *Windows* le volume des sons à jouer.

[4] Les développeurs de *MBROLA* à Mons nous ont annoncé qu'une librairie Windows offrant les mêmes fonctionnalités que le programme serait disponible sous peu.
[5] *MBROLA* propose plusieurs bases de données correspondant chacune à une langue et à une voix (masculine ou féminine) données. Les différentes langues disponibles incluent pour l'instant le français, l'anglais, l'allemand, l'espagnol et le portugais (brésilien).
[6] *MBROLA* utilise la technique des diphones pour composer les fichiers sonores, d'où la taille des bases de données disponibles (environ 5 Méga-octets chacune)

La création des fichiers sonores

Pour fonctionner, le programme *MBROLA* nécessite l'existence d'un fichier texte dans lequel se trouvent les phonèmes à générer dans le fichier sonore résultat. Le format précis de ce fichier texte se trouve dans les documents qui accompagnent le programme. En bref, il s'agit de placer sur chaque ligne du fichier la représentation d'un phonème suivie de la durée attendue pour ce phonème et éventuellement de la (ou des) hauteur(s) de ton à placer sur le phonème.

La première étape dans notre procédure sera donc de créer le fichier texte à passer au programme *MBROLA*. Pour ce faire, la représentation de chaque phonème demandé par l'utilisateur sera écrite dans le fichier, accompagnée de la durée moyenne du phonème considéré (voir annexe 1).

En ce qui concerne la hauteur de ton à placer sur le phonème, elle sera automatiquement calculée par la procédure en fonction de la longueur de la phrase phonétique et de l'intonation que l'utilisateur souhaite donner à sa phrase (intonation indiquée par un caractère spécial inclu dans la suite de phonèmes). L'utilisateur soucieux de personnaliser sa phrase pourra le faire en introduisant manuellement une hauteur de ton chiffrée (issue d'une échelle allant de 0 à 9).

Le fichier texte ainsi généré sera passé au programme *MBROLA* à l'aide d'une commande DOS. Notons que, pour réaliser cette commande, la procédure devra connaître également la base de données à utiliser (langue et sexe de la voix à utiliser) ainsi que le nom du fichier sonore attendu comme résultat de l'application du programme *MBROLA*.

La prononciation des fichiers sonores

Rien de compliqué ici lorsque l'on connaît l'existence de la *unit* pascal *MMSystem*. Cette *unit* implémente un certain nombre de fonctions destinées à la gestion de la carte son sous *Windows*. La fonction *SndPlaySound*, par exemple, joue automatiquement le fichier reçu comme paramètre.

Le règlage du volume sonore

A nouveau, la *unit MMSystem* va nous être d'un grand secours. Elle offre une fonction nommée *waveOutSetVolume* qui règle immédiatement le volume sonore sous *Windows* en fonction du paramètre qui lui est passé.

4.3.2 Code de la *unit*

Le code de la *unit MotVoc* se trouve intégralement dans l'annexe 4.

4.3.3 Difficultés rencontrées

Dans cette partie du développement du programme, trouver un moteur vocal sur le marché fut la chose la plus difficile. Malgré la grande quantité d'informations disponibles sur INTERNET, il nous a fallu plusieurs semaines pour découvrir que le site de la Faculté Polytechnique de Mons offrait les informations que nous cherchions. Il en fut de même en ce qui concerne les autres moteurs trouvés (pour la plupart inadaptés à notre projet).

Une fois le programme *MBROLA* découvert, l'implémentation de la *unit MotVoc* ne nécessita que le temps d'adaptation aux caractéristiques techniques de *MBROLA*.

CHAPITRE 5
LES STRUCTURES DE DONNEES

Dans notre programme, un certain nombre d'éléments sont paramétrables et se modifient d'une utilisation du programme à l'autre. Ces éléments peuvent provenir aussi bien de choix ou de modifications demandées par l'utilisateur que du fonctionnement propre du programme.

Ces éléments sont, d'une part, la structure de groupes créée par l'utilisateur et, d'autre part, les différents paramètres du programme (type de voix, type d'interface...). Il est nécessaire que ces deux structures de données soient disponibles tout au long de l'utilisation du programme, de même qu'elles doivent pouvoir être conservées d'une utilisation du programme à l'autre.

5.1 La structure de groupes

Cette notion de structure de groupes provient du chapitre 1, dans lequel nous avions proposé d'accélérer le débit de prononciation en mettant en place un système de préfabrication de phrases. Ces phrases, regroupées dans des groupes (également appelés contextes) interreliés donnent naissance à ce que nous nommons ici la structure de groupes.

Nous commencerons par décrire le format de cette structure, pour expliquer ensuite de quelle manière elle est implémentée et pour clôturer ce point sur les difficultés que nous avons rencontrées dans cette partie du programme.

5.1.1 Format de la structure

Notre structure sera donc constituée de groupes et de phrases.

Les phrases

Les phrases de la structure auront comme seule caractéristique un texte définissant le contenu de la phrase. Ce texte sera composé par la suite des caractères phonétiques que l'utilisateur souhaitera entendre lorsqu'il demandera la prononciation de la phrase.

Deux phrases distinctes ne pourront pas avoir le même texte.

Les groupes

Les groupes de la structure auront deux caractéristiques :
- Un nom, permettant de les différencier par rapport à tous les autres groupes;
- Une liste de liens, chaque lien référençant soit une phrase soit un autre groupe. Notons que, dans la structure, les références entre groupes devront toujours être réciproques. Si un groupe en référence un autre, ce second groupe devra obligatoirement référencer également le premier.

La structure

La structure sera donc composée de l'ensemble des groupes et des phrases existants.

Elle devra pouvoir donner un certain nombre de renseignements sur son contenu. Ces renseignements sont :
- la liste des groupes;
- la liste des phrases;
- la liste des groupes liés à un certain groupe;
- la liste des groupes non liés à un certain groupe;
- la liste des phrases liées à un certain groupe;
- la liste des phrases non liées à un certain groupe;
- la liste des groupes effacés par l'utilisateur;
- la liste des phrases effacées par l'utilisateur.

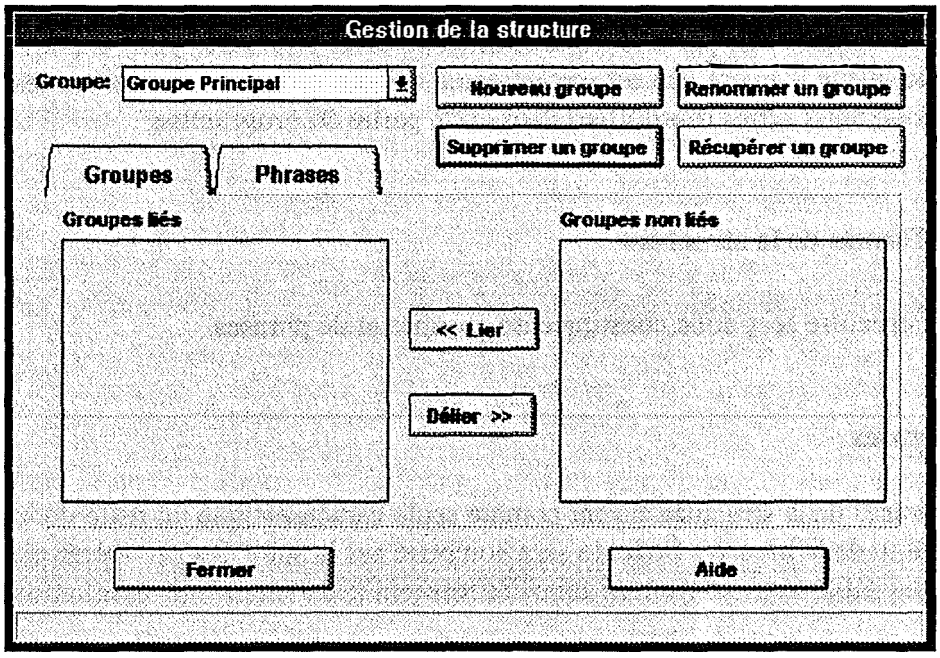

Figure 5.1

Elle devra également être en mesure de fournir un certain nombre de services de gestion des groupes et des phrases à l'intérieur de la structure. Ces services sont :
- la création et la destruction de la structure;

- l'ajout d'un groupe dans la structure;
- l'ajout d'une phrase dans la structure;
- la suppression d'un élément (groupe ou phrase) de la structure;
- le changement du texte d'une phrase ou du nom d'un groupe;
- la récupération d'un élément effacé par l'utilisateur;
- la liaison d'une phrase avec un groupe ou de deux groupes entre eux;
- la suppression d'un lien entre deux éléments.

La gestion de la structure devra pouvoir être proposée à l'utilisateur et devra se faire par le biais de la boîte de dialogue décrite dans la figure 5.1. Une fenêtre identique, accessible via les onglets disponibles dans la boîte, permettra de gérer les phrases[7].

On peut remarquer, dans le bas de la boîte, la présence d'une zone vide destinée à accueillir du texte. Cette zone de texte permettra au programme d'afficher de l'aide pour l'utilisateur. Par exemple, lorsque le curseur de la souris sera placé au dessus de l'un des boutons de la boîte, le programme affichera un texte expliquant l'usage de ce bouton.

5.1.2 Choix de la représentation interne de la structure

Nous désignerons les groupes et les phrases de la structure sous le terme générique d'élément. Afin de stocker les éléments, nous allons définir un objet *TStruct* héritant des propriétés de l'objet pascal *TCollection*.

Les éléments

Tous les éléments de la structure ont des caractéristiques communes que nous pouvons regrouper dans un objet du type *TElement* (que nous ferons hériter de *TObject*). Cet objet servira d'ancêtre aux différents acteurs de notre structure de groupes. Les champs dont il sera pourvu et qui serviront autant aux groupes qu'aux phrases sont

- un champ permettant de différencier les groupes des phrases;
- un *String*, qui jouera le rôle de nom pour les groupes et de texte pour les phrases;
- un booléen permettant de définir, pour chaque élément, un état *effacé* et un état *non effacé*. Grâce à ce booléen, l'utilisateur pourra demander l'effacement d'une phrase ou la suppression d'un groupe puis, par la suite, en cas d'erreur par exemple, demander la récupération du groupe ou de la phrase dans l'état ou il/elle était lors de la suppression.

Aucune méthode particulière ne devra être créée pour l'objet *TElement*, toutefois, il sera nécessaire de redéfinir les méthodes *Init* et *Done* de l'ancêtre *TObject* et de

[7] La gestion des phrases se fera de la même manière que la gestion des groupes

définir les méthodes *Load* et *Store* pour la lecture et l'écriture dans un flux (pour la sauvegarde de l'objet en tant qu'élément de la structure complète).

Les phrases

En ce qui concerne les phrases de la structure, elles n'ont aucune caractéristique de plus que les objets de type *TElement*. Ce type sera donc celui utilisé pour les définir.

Les groupes

Les groupes, quant à eux, nécessitent certains champs dont nous n'avons pas encore parlé. Nous allons donc définir un nouveau type d'objet (*TGroupe*) destiné à décrire les groupes de la structure. Bien entendu, *TGroupe* héritera de *TElement*.

Les champs propres à *TGroupe* sont
- un pointeur vers une collection de références, chaque référence comportant le nom d'un autre élément de la structure ainsi qu'un pointeur vers cet élément. C'est grâce à cette collection qu'il sera possible de définir des liens entre les groupes et entre les groupes et les phrases. Une méthode de la structure permettra, pour un groupe donné, de retrouver les groupes ou les phrases qui y sont liés ainsi que de gérer l'ajout ou la suppression de liens entre éléments;
- un booléen indiquant si l'effacement du groupe est autorisé. De cette manière il sera possible de définir un groupe principal que l'utilisateur pourra renommer mais pas supprimer et qui assurera à tout moment l'existence d'au moins un élément dans la structure. Cet élément n'est pas obligatoire mais il concourt à la cohérence de la structure.

TGroupe redéfinira les méthodes *Init*, *Done*, *Load* et *Store* de son ancêtre *TElement*.

La structure

L'objet *TStruct*, comme cela a été spécifié plus haut, possédent un certain nombre de fonctionnalités; le but de ces fonctionnalités étant de permettre une consultation et une gestion plus aisée et plus rapide des éléments contenus dans la structure.

Les services de consultations disponibles sont :
- *ListeGrp*, la liste des groupes de la structure;
- *ListePhr*, la liste des phrases de la structure;
- *ListeGrpLies*, la liste des groupes liés à un certain groupe de la structure;
- *ListeGrpNonLies*, la liste des groupes non liés à un certain groupe de la structure;
- *ListePhrLies*, la liste des phrases liées à un certain groupe de la structure;
- *ListePhrNonLies*, la liste des phrases non liées à un certain groupe de la structure;

- *ListeGrpEff*, la liste des groupes effacés par l'utilisateur;
- *ListePhrEff*, la liste des phrases effacées par l'utilisateur.

Tous ces services renvoient une collection contenant des pointeurs vers les éléments correspondant au résultat de la requête.

Les services de gestion, quant à eux, sont :

- *Init* et *Done*, la création et la destruction de la structure;
- *Load et Store*, la sauvegarde et la récupération de l'objet dans un flux;
- *AjoutGrp*, l'ajout d'un groupe dans la structure;
- *AjoutPhr*, l'ajout d'une phrase dans la structure;
- *SupprEl*, la suppression d'un élément (groupe ou phrase) de la structure;
- *RenEl*, le changement du texte d'une phrase ou du nom d'un groupe;
- *RecupEl*, la récupération d'un élément effacé par l'utilisateur;
- *LierEl*, la liaison d'une phrase avec un groupe ou de deux groupes entre eux;
- *DelierEl*, la suppression d'un lien entre deux éléments.
- *Gest*, l'exécution de la boîte de dialogue de gestion de la structure (accès aux fonctionnalités décrites ci-dessus).

5.1.3 Code de la *unit*

Le code complet de la *unit GestGrp* implémentant les objets décrits ci-dessus se trouve dans l'annexe 5.

5.1.4 Difficultés rencontrées

Face à certaines difficultés rencontrées pour développer cette unit, un certain nombre de choix ont dû être faits. Nous proposons de décrire ici ces choix et leur raison.

La première difficulté fut d'imaginer de quelle manière représenter la structure. Ne connaissant pas l'objet *TCollection*, nous étions partis dans la conception d'une liste de pointeurs bien complexe (nous étions en fait en train d'inventer notre propre *TCollection* !). Sur les conseils du professeur Cherton, nous nous sommes orientés vers la solution simple et efficace (notamment grâce à l'existence des méthodes *ForEach* et *FirstThat*) de l'utilisation des collections pascal.

Un autre problème que nous avons rencontré fut la sauvegarde de la liste de liens des groupes. En effet, si nous avions sauvegardé les références aux éléments (c'est à dire le pointeur et le contenu de l'objet vers lequel il pointe), ces éléments se seraient trouvés sauvegardés plusieurs fois : une fois avec tous les objets de la structure et une fois pour chaque pointeur les référençant. De plus, le problème des références en boucle (groupes se référençant mutuellement) auraient entraîné des situations de bouclage des procédures de sauvegardes. Il n'était donc pas concevable de sauvegarder telle quelle la liste de liens de chaque groupe.

C'est pourquoi nous avons transformé la liste de liens de chaque groupe. Chaque lien est devenu une référence comportant le nom (le texte) du groupe (de la phrase) référencé(e) et un pointeur vers cet élément. Lors du sauvetage dans un fichier, seul le nom est enregistré. Lors de la lecture dans le fichier (récupération des données), les listes de références sont reconstituées en faisant pointer le pointeur de chaque référence vers l'élément dont le nom correspond au nom sauvegardé. Bien entendu, ceci signifie que le champ *Nom* de chaque élément doit devenir identifiant de cet élément. Il sera nécessaire, lors de la création de chaque nouvel élément, de s'assurer que cette propriété est vérifiée.

5.2 Les paramètres utilisateurs

Au cours de l'utilisation du programme, il y a un certain nombre de choses dont l'utilisateur peut souhaiter la modification (apparence de l'interface, type de voix de prononciation...). De la même manière, le programme lui-même peut être amené à changer la valeur de certaines variables environnementales (Nom du groupe en cours de gestion...).

La différence principale entre ces variables (que l'on nommera ici paramètres) et des variables classiques de programmation (que l'on trouve également dans notre logiciel) est qu'il est important de conserver la valeur de ces paramètres d'une utilisation du programme à l'autre. Le but de cette conservation est que l'utilisateur retrouve le programme dans l'état où il l'a laissé.

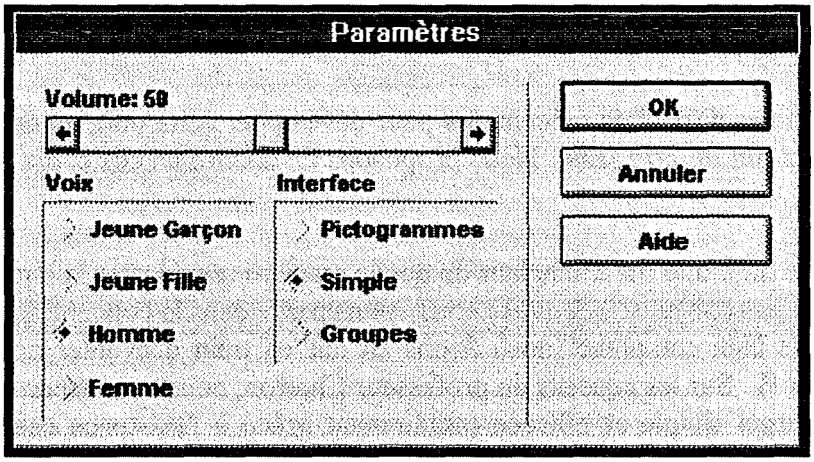

Figure 5.2

5.2.1 Paramètres utiles

Les différents éléments à conserver d'une utilisation du programme à l'autre sont :
- Le type d'interface;
- Le type de voix;
- Le volume de prononciation;
- Les noms des groupes présentés dans l'interface de présentation des groupes;

- Le nom du dernier groupe géré dans la boîte de dialogue de gestion de la structure de groupes.

Le programme devra également pouvoir proposer une boîte de dialogue de gesion de ces paramètres. Cette boîte est visible dans la figure 5.2.

S'il n'est pas possible de modifier les deux derniers typs de paramètres à partir de cette boîte, c'est parce qu'ils sont mis à jour automatiquement durant l'utilisation du programme.

5.2.2 Choix de la représentation interne des paramètres

Nous avons regroupé les paramètres dans un nouvel objet du type TParam. Les différents champs de l'objet *TParam* constitueront les paramètres.

On retrouve dans l'objet :
- Un champ décrivant le type d'interface choisi par l'utilisateur (simple, à pictogrammes ou présentant la structure de groupes);
- Un champ décrivant le type de voix (d'adulte ou d'enfant, masculine ou féminine);
- Un *Integer* indiquant le volume de prononciation tel que réglé par l'utilisateur (valeur entre 0 et 100);
- Deux *Strings* permettant d'enregistrer le nom des groupes affichés dans la fenêtre principale de présentation de la structure de groupes;
- Un *String* permettant de conserver le nom du groupe en cours de gestion dans la fenêtre de gestion des groupes et des phrases;
- Deux *Integers* définissant le numéro du premier fichier prononcé au cours de cette session et le numéro du prochain fichier à prononcer;
- Un *Integer* définissant le numéro du prochain fichier à prononcer dans la fenêtre de gestion des groupes et des phrases.

En ce qui concernent les méthodes de l'objet, rien de particulier ne s'est avèré utile. On a simplement défini des méthodes *Init*, *Done*, *Load* et *Store* propres permettant la gestion de base de l'objet.

Une méthode *Gest* donne accès à la fenêtre de gestion des paramètres.

5.2.3 Code de la *unit*

Le code complet de la *unit GestPar* implémentant les objets décrits ci-dessus se trouve dans l'annexe 6.

5.2.4 Difficultés rencontrées

Dans les premières versions du programme, les paramètres étaient regroupés dans un record pascal. La raison de ce choix était la simplicité du sauvetage d'un record dans un fichier (il est possible d'écrire l'entièreté d'un record en une opération) ainsi que sa simplicité d'utilisation. Toutefois, en raison de l'Orientation Objet donnée à l'entièreté du programme Cyrano, il nous a semblé plus normal de transformer ce record en objet.

Une autre difficulté est apparue lors des tests du programme. Les fonction de la *unit MMSystem*[8] que nous utilisons pour jouer des fichiers sonores dans notre moteur vocal sont un peu particulières en ce sens qu'elle se souviennent de tout fichier sonore qu'elles ont joué. Si vous demandez à *MMSystem* de jouer le fichier nommé *texte.wav*, elle le jouera. Si ensuite vous faites subir une modification au fichier *texte.wav* et que vous demandez à nouveau à *MMSystem* de la jouer, vous n'entendrez pas les modifications effectuées mais uniquement le fichier originel[9].

Par conséquent, il est nécessaire que le nom du fichier sonore soit différent à chaque fois que l'on demande une prononciation. C'est pourquoi nous avons décidé de donner un numéro à ce fichier et d'incrémenter le numéro lors de chaque demande de prononciation. De plus, pour pouvoir quitter le programme puis y revenir dans la même session *Windows*, il fallait que les numéros des fichiers de la seconde exécution du programme soient différents des numéros de la première. C'est la raison pour laquelle le numéro du fichier suivant à créer lors de la prononciation se retrouve parmi les paramètres du programme à enregistrer.

Le fait que chaque fichier sonore porte un nom différent des autres a une conséquence avantageuse. Etant donné que nous n'effaçons les fichiers sonores du disque où ils ont été écrits qu'à la fin de l'exécution du programme, il est possible de les jouer plusieurs fois au cours de la même exécution. Il suffit de repasser à la *unit MMSystem* le nom du fichier et la prononciation a lieu quasiment instantanément. On évite, en effet, tout le temps de calcul pris par *MBROLA* puisqu'il n'est plus nécessaire d'y faire appel.

[8] *Unit* fournie avec Borland Pascal 7.0.

[9] Sans doute un système de cache implémenté par les fonctions de la *unit* est-il à l'origine de ces problèmes.

CHAPITRE 6
LE PROGRAMME PRINCIPAL

Véritable épine dorsale de notre application, le programme principal coordonne toute l'interaction entre les différents éléments logiciels (la fenêtre principale, les structures de données...). Nous décrirons ici tous les éléments qui n'ont pas fait l'objet d'un des chapitres précédents, à savoir le programme principal lui-même et la fenêtre principale ainsi que les petites *units* accessoires qu'il a été nécessaire de mettre en place.

6.1 Le programme principal

Nous allons commencer par décrire les comportements attendus de la part du programme principal, pour voir ensuite sous quelle forme ces comportements vont être implémentés.

6.1.1 Comportements attendus

Lors du lancement du programme, celui-ci devra effectuer les différentes opérations suivantes :

- Mise en place les différentes structures de données décrites au chapitre 5;
- Choix pour l'affichage de la fenêtre principale de l'interface adaptée en fonction des choix faits par l'utilisateur (paramètres);

En fin d'exécution, le programme devra effectuer les opérations suivantes :

- Sauvegarde des structures de données (chapitre 5);
- Effacement des fichiers temporaires créés par le programme (*.Wav* et *.Pho*).

6.1.2 Implémentation de l'application

Nous avons créé un objet héritant de *TApplication*. Dans cet objet, sont redéfinies les différentes méthodes servant à initialiser et à clôturer l'application. La méthode *Run* (de *TApplication*) n'est, quant à elle, pas redéfinie car il n'était pas nécessaire de lui faire subir des modifications. En effet, son fonctionnement actuel - l'exécution de la fenêtre désignée par le champ *MainWindow* - est suffisant pour l'utilité que nous en avons.

Initialisation globale (Init)

Dans cette phase du programme principal (la première), différents éléments utilisés par l'application sont mis en place. Les curseurs particuliers au programme sont créés et leurs *Handles* sont placés dans des variables globales. Les structures de données (paramètres et groupes) sont mises en place (en faisant appel aux méthodes adéquates décrites dans le chapitre 5).

Initialisation de la fenêtre principale (InitMainWindow)

Le choix de l'interface à afficher dès le lancement du programme dépend de la valeur du champ *TypeInterf* de la variable globale *Param* (référençant les paramètres utilisateurs).

Nous commençons donc par initialiser trois pointeurs vers les trois types de fenêtres possibles, pour ensuite, en fonction de la valeur du champ *TypeInterf*, définir adéquatement le champ *MainWindow* de notre application. En ce qui concerne les deux pointeurs qui n'ont pas été utilisés, il faut simplement faire appel à leur méthode *Create* pour que, par la suite, les fenêtres qu'ils référencent puissent être affichées en remplacement de la fenêtre principale si l'utilisateur le demande.

Initialisation de l'instance de l'application (InitInstance)

Ici, les accélérateurs qui se trouvent dans les ressources créées pour le programme sont chargés.

Destruction de l'application (Done)

Dernière méthode appelée lors de l'exécution du programme, elle fait le ménage avant de rendre la main. Dans notre cas, elle détruit le curseur de l'application (*hautparleur*); fait appel aux destructeurs des structures de données (voir chapitre 5); élimine tous les fichiers temporaires (*.Wav* et *.Pho*) créés lors de l'exécution du programme; puis enfin fait appel à sa méthode ancêtre (*TApplication.Done*).

6.1.3 Code du programme principal

Le code complet du programme principal (fichier *Cyrano.pas*) se trouve dans l'annexe 2.

6.2 La fenêtre principale

Nous allons maintenant décrire les trois interfaces possibles du programme ainsi que leurs fonctionnalités. Pour ce faire, nous commencerons par la fenêtre la plus

simple, pour passer ensuite à celles présentant les pictogrammes et la structure de groupe.

6.2.1 Interface simple

Cette fenêtre sera destinée à offrir tous les services de base de l'application; à savoir :

- prononciation en cliquant sur un bouton marqué d'un haut parleur d'une phrase phonétique tapée dans un champ d'édition;
- prononciation d'une des phrases précédemment prononcées, présentées dans une liste de sélection;
- réglage du volume de prononciation, via une barre de défilement verticale;
- appel d'une fenêtre d'aide du programme, grâce à un bouton marqué d'un point d'interrogation.

Nous organiserons ces différents services dans la fenêtre de la manière indiquée dans la figure 6.1.

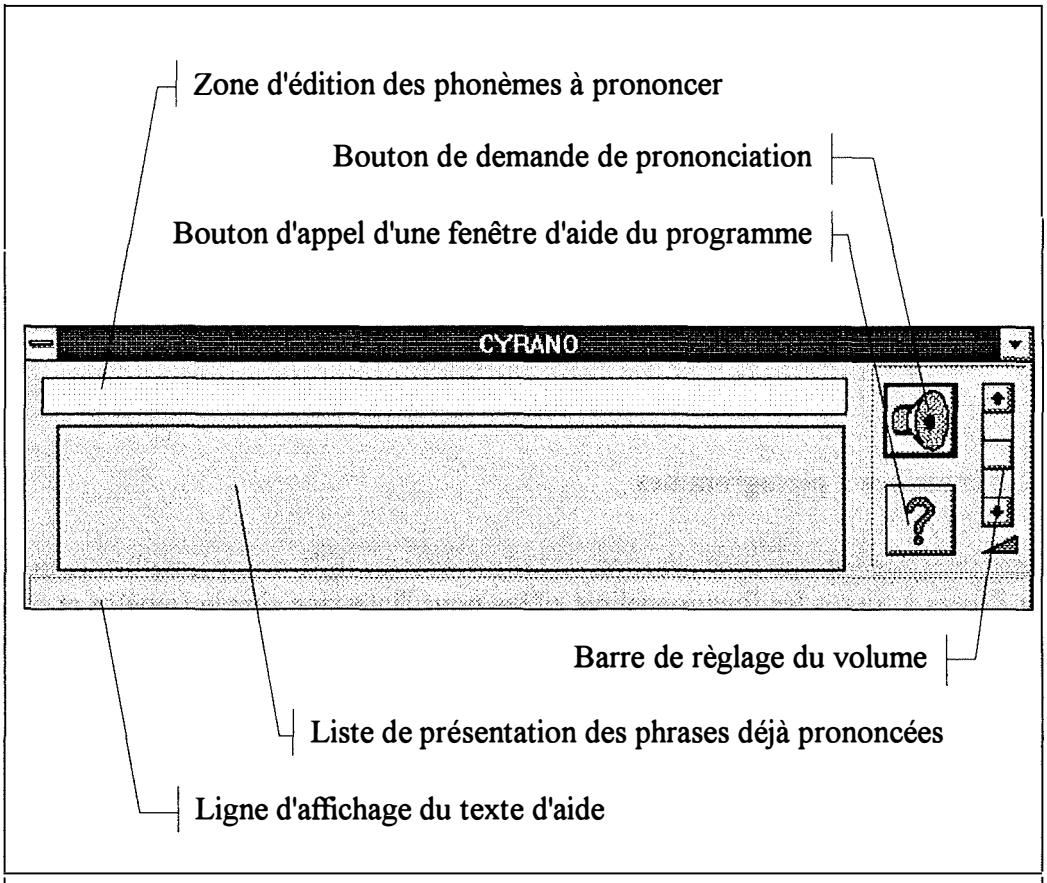

Figure 6.1 : Interface simple

Notons ici l'utilisation un peu particulière de la souris dans cette interface. En ce qui concerne la liste présentant les phrases déjà prononcées au cours de la session, l'utilisateur peut effectuer deux actions à l'aide de la souris. Il peut, s'il le souhaite, demander de reprononcer une phrase en cliquant sur cette phrase avec le bouton

droit de la souris (prononciation immédiate). Il peut également demander que la phrase soit placée dans le champ d'édition de la fenêtre (pour pouvoir la modifier par exemple). Cette seconde commande est accessible en cliquant sur la phrase avec le bouton gauche de la souris.

Cette ergonomie, pouvant paraître un peu étonnante au premier abord, a été imaginée pour respecter la configuration gauche/droite de l'écran. En haut à gauche de la fenêtre se trouve le champ d'édition des phrases phonétiques. Le bouton gauche de la souris est donc lié à ce champ d'édition. En haut à droite de la fenêtre se trouve le bouton de prononciation des phrases phonétiques. Le bouton droit de la souris est donc lié à la prononciation immédiate des phrases.

A cette fenêtre sera adjoint un menu, offrant toutes les commandes générales disponibles pour l'application. Ces différentes commandes permettront de demander

- la prononciation d'une phrase éditée;
- la gestion des paramètres;
- l'impression dans le bloc-notes *Windows* du résumé des phrases prononcées au cours de la session d'utilisation courante;
- la gestion des groupes et des phrases;
- la transformation d'une phrase éditée en phrase préfabriquée;
- l'affichage d'une fenêtre d'aide sur le programme.

Implémentation

L'interface simple sera implémentée dans un objet du type *TCyrDlg*. Cet objet, défini comme héritant du type d'objet *TDlgWindow*, sera initialisé avec la boîte de dialogue décrite dans la figure 6.1. Ses différentes méthodes réagiront aux actions effectuées par l'utilisateur dans la boîte de dialogue.

6.2.2 Interface à pictogrammes

Cette interface est une spécialisation de la précédente. Elle doit permettre de donner l'accès à toutes les fonctionnalités citées pour l'interface simple (fenêtre et menu) et doit en plus offrir un certain nombre de boutons associés aux phonèmes existants.

Lorsque l'utilisateur souhaite écrire un phonème dans le champ d'édition, cette interface doit lui donner le choix de le faire soit au clavier (en poussant sur le caractère correspondant au phonème) soit en cliquant à l'aide de la souris sur le bouton associé au phonème.

Par rapport à ce qui a été fait au point précédent, il n'y a donc qu'à ajouter le clavier phonétique virtuel de la méthode PHONEPIC sur l'écran. Cela donne la fenêtre que l'on peut voir dans la figure 6.2. On remarque que la partie supérieure de la figure est identique à celle de la figure 6.1 et offre donc les mêmes fonctionnalités.

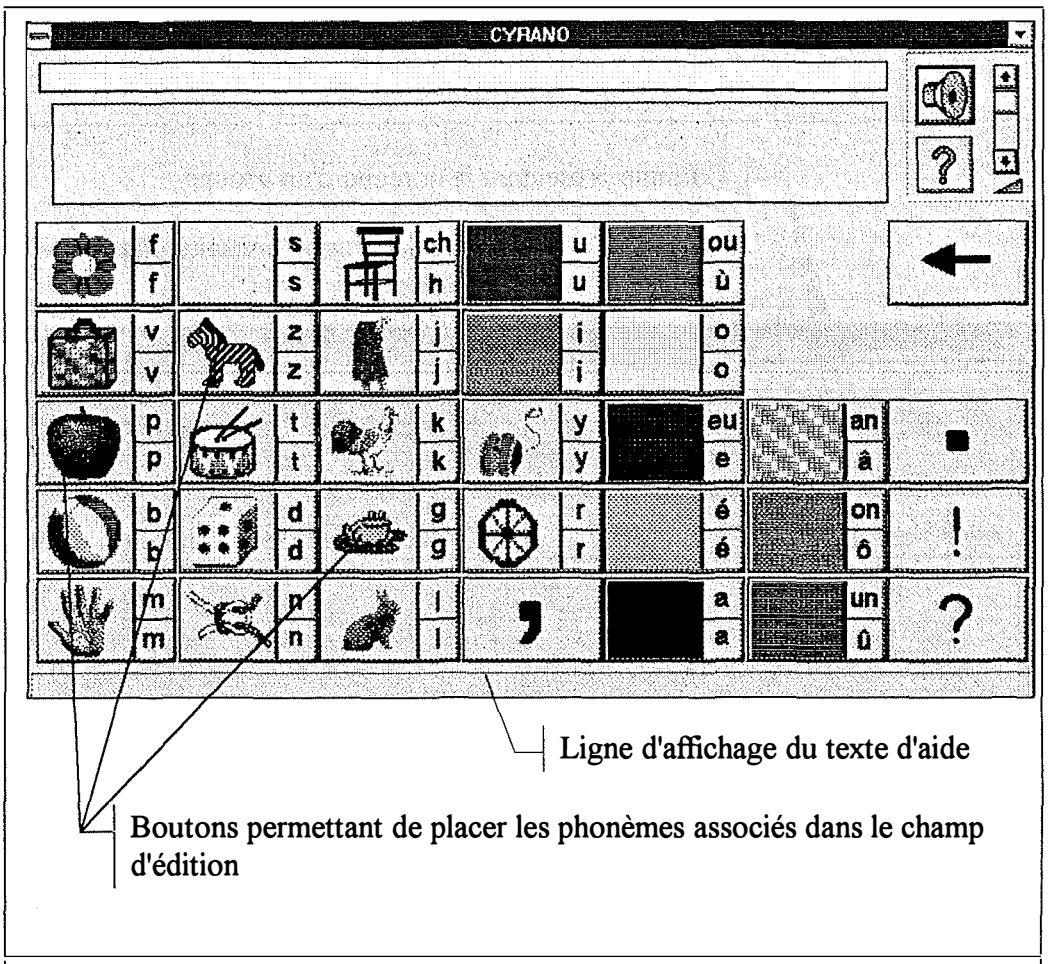

Figure 6.2 : Interface à pictogrammes

Implémentation

Nous avons défini un objet du type *TCyrDlgPicto* héritant de *TCyrDlg*. Les boutons de l'interface ont été redéfinis à partir du type pascal *TButton*. Ces boutons ont été implémentés pour répondre à un clic de la souris en remplissant le champ d'édition de la fenêtre avec un caractère donné. L'objet *TCyrDlgPicto* implémente les méthodes d'initialisation et de mise en place nécessaires au fonctionnement du clavier à pictogrammes.

6.2.3 Interface présentant la structure de groupes

De la même manière que l'interface à pictogrammes, celle-ci est une spécialisation de l'interface simple.

Cette interface aura un certain nombre de caractéristiques supplémentaires par rapport à l'interface simple. Toutes ces nouvelles caractéristiques vont concerner la possibilité de se déplacer dans la structure de groupes et d'y sélectionner des phrases à prononcer. Ainsi, nous allons intégrer dans la fenêtre la possibilité de visualiser le contenus d'un groupe (groupes liés et phrases liées). Grâce à la souris, l'utili-

sateur pourra alors choisir le groupe dont il souhaite voir apparaître le contenu ou la phrase qu'il souhaite prononcer.

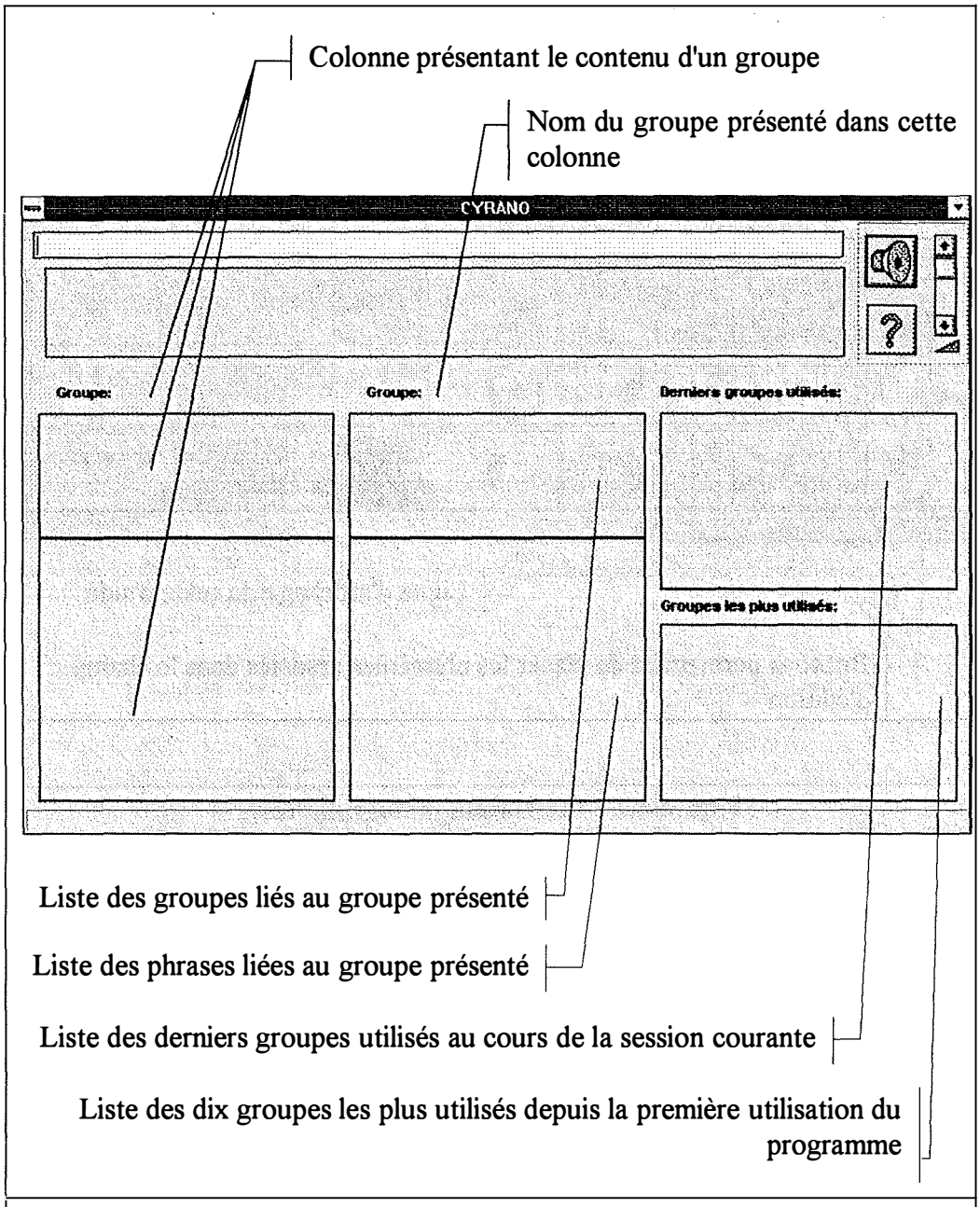

Figure 6.3 : Interface présentant la structure de groupes

Nous agencerons les différents éléments dans la fenêtre de la manière décrite dans la figure 6.3. A nouveau, on retrouve, dans la partie supérieure de la figure, les mêmes éléments que dans les figures 6.1 et 6.2. En ce qui concerne les éléments nouveaux, leur utilité et leur fonctionnement sont décrits dans ce qui suit.

Les deux colonnes de gauche servent à présenter chacune le contenu d'un groupe. La liste de sélection supérieure servant à afficher les noms des groupes liés au

groupe présenté et la liste inférieure servant à afficher le texte des phrases liées au groupe présenté.

L'utilité des deux listes de droite est d'afficher automatiquement la liste de tous les groupes utilisés au cours de la session courante[10] ainsi que celle des dix groupes les plus utilisés depuis la première utilisation du programme.

De la même manière que pour la manipulation des phrases dans la liste de sélection des phrases déjà prononcées (voir plus haut), l'ergonomie de l'utilisation de la souris est un peu particulière.

Dans les listes affichant des textes de phrases, l'utilisation de la souris est la même que dans la liste de sélection des phrases déjà prononcées (bouton gauche/champ d'éditon - bouton droit/prononciation).

Dans les listes affichant des noms de groupes, le bouton gauche de la souris sert à placer dans la colonne de gauche le contenu du groupe placé sous le pointeur de la souris. Le bouton droit de la souris sert, quant à lui, à placer dans la colonne de droite le contenu du groupe pointé par la souris. On voit que, ici encore, la répartition gauche/droite de l'écran a été respectée (bouton gauche/colonne gauche - bouton droit/colonne droite).

Implémentation

Nous avons défini un objet du type *TCyrDlgGroupes* héritant de *TCyrDlg*. *TCyrDlgGroupes* implémente les méthodes nécessaires au fonctionnement de l'interface. La plupart des *listbox* présentes dans la fenêtre ont été redéfinies pour avoir un comportement propre réalisant la fonctionnalité pour laquelle elle sont mises en place. Les seules fonctionnalités définies au niveau de l'objet *TCyrDlgGroupes* sont la mise en place et l'initialisation de la fenêtre ainsi que la mise à jour de la fenêtre.

6.2.4 Code de la fenêtre principale

Le code complet des trois interfaces se trouve dans l'annexe 3.

6.3 *Units* annexes

Remarquons ici l'existence de deux petites *units* que nous avons dû créer pour le bon déroulement du programme mais qui ne nécessitent pas tout un chapitre d'explications.

[10] Les groupes sont placés dans la fenêtre dans l'ordre de leur utilisation. Si un groupe a été utilisé plusieurs fois au cours de la session, le nom de ce groupe est affiché uniquement à la position correspondant à son utilisation la plus récente

6.3.1 La *unit CyrTyp*

La première, *CyrTyp.tpw*, implémente un ensemble de types d'objets dérivés des types windows. A ces objets, nous avons donné des caractéristiques supplémentaires utiles, la plus importante étant la réponse à un passage de la souris au dessus de l'objet. Lorsqu'un objet détecte la présence de la souris au dessus de lui, il envoie, à un objet de type *TStatic* déterminé, un *String* à afficher.
Cette fonctionnalité est utilisée pour implémenter un système de ligne d'aide dans la fenêtre principale. Lorsque la souris se déplace au dessus de la fenêtre principale, l'utilité de l'élément au dessus duquel la souris se trouve est décrite dans une ligne de texte au bas de l'écran.

Il va sans dire que les types d'objets implémentés dans cette *Units* sont les types d'objets d'interaction avec l'utilisateur dont nous avons eu besoin pour construire nos fenêtres.

6.3.2 La *unit VarGlob*

La seconde *Unit* est la *Unit VarGlob.tpw*. Dans cette *Unit* sont définies les variables globales du programme. De cette manière, tout module du programme peut avoir accès aux variables globales en incluant simplement dans sa clause *Uses* le nom de cette *Unit*. Les variables accessibles de cette manière sont les curseurs propres au programme et les pointeurs vers les paramètres utilisateurs et la structure de groupe.

6.3.3 Code des *units*

Les codes complets des deux *units* (fichiers *VarGlob.pas* et *CyrTyp.pas*) se trouvent dans l'annexe 7.

6.4 Difficultés rencontrées

Dans cette partie du logiciel, la difficulté est survenue de la méthode choisie pour implémenter la triple interface. Nous avons opté (par un manque de connaissance d'autres solutions possibles) pour une solution peu commode à gérer.

Pour commencer, nous avons créé les trois objets dérivés de *TCyrDlg* (*TCyrDlg*, *TCyrDlgPicto* et *TCyrDlgGroupes*) puis nous avons choisi dynamiquement l'objet recevant le statut de *MainWindow* en fonction des choix de l'utilisateur.

En cours d'utilisation, il nous a fallu nous assurer que chaque interface pouvait passer la main à n'importe quelle autre (c'est à dire que chaque interface connaissait la référence des deux autres). Ce fut réalisé en passant, dès l'initialisation des objets, trois pointeurs vers les trois objets héritant de *TCyrDlg*. De cette manière, il

était toujours loisible à l'une des trois interfaces de s'adresser à une autre (par exemple pour lui passer la main). Il fallait également s'assurer que certains messages étaient envoyés vers les trois interfaces simultanément (par exemple les messages de remplissage de la liste de sélection des phrases déjà prononcées).

Pour finir, au moment de la destruction des trois objets (juste avant la destruction de l'application elle-même), il nous a fallu nous assurer de la destruction de l'interface courante en dernier lieu pour ne pas perdre dans la mémoire les deux autres interfaces. Ceci fut fait en créant une destruction à deux niveaux : le premier envoyant aux deux interfaces cachées l'ordre de se détruire et le second détruisant effectivement l'interface courante.

Une autre difficulté est apparue lorsque monsieur Lamy nous a fait remarquer (lors d'une présentation de l'avancement du travail) qu'il n'était pas possible à un utilisateur de se servir du clavier à pictogrammes pour créer des phrases de la structure de groupes. C'est la raison pour laquelle nous avons introduit la commande de menu *En faire une phrase*.

L'utilisateur peut maintenant éditer, dans la zone d'édition de texte de la fenêtre principale, une phrase au moyen des pictogrammes (ou du clavier) puis transformer directement cette phrase éditée en phrase préfabriquée grâce à la commande de menu *En faire une phrase*. Cette nouvelle phrase préfabriquée est automatiquement liée avec le *groupe principal* de la structure.

CHAPITRE 7
EVALUATION DU PROGRAMME FINAL

7.1 L'avis de Monsieur Lamy

Lorsque nous avons montré la version finale du programme à monsieur Lamy, il a paru plutôt satisfait par le résultat. Dans l'ensemble, le programme a semblé bien correspondre aux attentes qu'il avait.

Il a toutefois fait un certain nombre de remarques sur les caractéristiques que n'a pas le programme et qui auraient pu être intéressantes.

Le caractère d'aération du texte

La première remarque concerne la difficulté de lire les phrases phonétiques (tapées dans le champ d'édition ou préfabriquées). Au delà d'une demi douzaine de caractères, le texte phonétique devient réellement difficile à déchiffrer.

En effet, dans le texte phonétique des phrases, le caractère blanc (espace) a une signification (pause au moment de la prononciation). On ne peut donc pas se servir de ce caractère pour rendre plus lisible la phrase éditée.

Monsieur Lamy a alors pensé à la définition d'un caractère particulier destiné à aérer le texte. Ce caractère serait visible dans les phrases mais n'aurait aucune signification phonétique et ne s'entendrait pas lors de la prononciation.

Nous avons choisi pour cela le caractère '_' (*underscore*). La conception actuelle du moteur vocal fait que, lorsqu'il rencontre un caractère inconnu, il l'ignore. Sans que nous le sachions, cette caractéristique était donc déjà présente dans le programme.

Dans la structure de groupe, ce caractère pourrait être automatiquement ajouté à la fin de chaque phrase préfabriquée. Ainsi, lorsque l'utilisateur souhaiterait introduire une phrase préfabriquée dans le champ d'édition, elle serait automatiquement séparée des autres caractères édités par le symbole '_'.

Grâce à ceci, on gagne en lisibilité (aération du texte), ce qui permet de se relire plus aisément avant de demander la prononciation d'une phrase.

Les moyens d'interaction

Une deuxième remarque de monsieur Lamy concerne les moyens d'interaction avec le programme. Le fait est que, pour utiliser adéquatement le programme, l'utilisateur est obligé de se servir de la souris ET du clavier.

Avec la souris, la seule chose que vous ne puissiez pas faire est de créer un groupe (le clavier est nécessaire pour introduire le nom du groupe).

Avec le clavier, les fonctions de base d'édition et de prononciation de phrases sont accessibles mais un grand nombre de fonctionnalités ne le sont pas.

Il aurait été intéressant que l'utilisation du programme ait pu se faire intégralement à l'aide de la souris OU du clavier. La raison en est simple. Il existe un certain nombre d'outils d'interaction qui ont été créés pour permettre à certains types de handicapés d'utiliser l'ordinateur. Pour les personnes utilisant un moyen d'interaction de ce type, ce dernier est souvent la seule manière de se servir d'un ordinateur. Or la particularité de ces outils est d'avoir été fabriqués pour remplacer exactement soit le clavier soit la souris (de manière à ce que l'ordinateur ne puisse pas faire la différence).

Si notre programme avait pu être intégralement manipulé à l'aide d'un seul moyen d'interaction (souris ou clavier), les outils d'interaction créés pour les handicapés auraient pu être utilisés pour se servir de notre programme.

Cependant, comme nous l'avons fait remarqué plus haut, la souris permet d'accéder à la presque totalité des caractéristiques du programme. La possibilité est donc toujours laissée à l'utilisateur d'utiliser un moyen d'interaction unique remplaçant la souris pour se servir de Cyrano; cette méthode empêchant malheureusement de créer de nouveaux groupes dans la structure.

Un programme multi-utilisateur

Une autre faiblesse de notre programme est de ne pas pouvoir servir à plusieurs personnes en même temps. La raison en est qu'il n'est ni possible de sauver deux paramétrages du système différents ni possible de sauver deux structures de groupes différentes.

Un utilisateur se trouve donc toujours confronté à l'état du programme tel qu'il était au moment de la fin de la dernière utilisation. S'il souhaite une configuration particulière du programme, il doit la redéfinir chaque fois qu'une autre personne a changé cette configuration. De la même manière, s'il veut une structure de groupe particulière, il doit la redéfinir chaque fois que la structure de groupe a été changée par l'utilisateur.

Il serait possible de remédier à cela sans toucher à Cyrano. Si nous créions un petit programme servant de gestionnaire de fichiers de configuration et lançant lui-même

le programme Cyrano, ce petit programme pourrait proposer différents noms d'utilisateurs, chacun associé à un nom de fichier de configuration différent.

Lorsque vous indiqueriez au programme quel est votre nom d'utilisateur il modifierait les fichiers *param.dat* et *struct.dat* avec lesquels travaille Cyrano pour qu'ils correspondent aux paramètres propres à l'utilisateur sélectionné. Lorsque le programme Cyrano se terminerait, le même programme, reprenant un instant la main, sauverait les fichiers *param.dat* et *struct.dat* dans un fichier de configuration portant le nom de l'utilisateur.

7.2 Les premiers essais

Malgré un manque de temps certain, il nous a été possible de faire essayer notre programme par un petit nombre de personnes (cinq) néophytes en informatique. Ces personnes n'étant pas muettes, il leur a été difficile de tester l'intérêt réel du programme

Dans tous les cas, ces personnes ont trouvé le programme assez pratique et facile à utiliser. Certaines remarques ont cependant été faites :

- L'apprentissage nécessaire à la manipulation des phonèmes est un processus long et pénible;
- Il est nécessaire d'avoir une bonne rapidité de frappe au clavier pour pouvoir utiliser pleinement le programme;
- Il y a une possibilité de confusion entre les différentes manières de prononcer des phrases : soit on édite la phrase (champ d'édition) et on pousse sur le bouton de prononciation; soit on clique directement sur une phrase (liste de sélection) avec le bouton droit de la souris. La tentation est grande de cliquer sur la souris après avoir édité une phrase dans le champ d'édition ou de pousser sur le bouton de prononciation pour entendre une phrase sélectionnée dans une liste de sélection;
- Certains phonèmes disponibles n'apparaissent pas dans l'interface à pictogrammes. Ce fait est déroutant et rend encore plus difficile l'apprentissage de la représentation des phonèmes;
- La notion de liens entre groupes est un peu confuse pour certaines personnes. Ces personnes en général confondent l'idée de chemin d'accès (qui est ce que représentent les liens entre éléments) avec l'idée de contenance (un groupe en *contiendrait* d'autres, d'où l'incompréhension du fait que des groupes distincts peuvent *contenir* tout deux un même troisième);
- Dans la fenêtre de gestion des groupes et des phrases, il y a une difficulté pour comprendre comment on gère les phrases. La gestion des groupes entre eux ne pose pas de problème (ajout ou suppression de liens entre groupes) alors que la gestion des phrases (qui est pourtant identique à celle des groupes) soulève des questions et des difficultés de compréhension. Peut-être la représentation mentale que les personnes ont de leur structure de groupes et de phrases rend-elle cette compréhension difficile.

Il est certain que ce n'est que lorsqu'un plus grand nombre de personnes auront testé le programme et que des personnes muettes auront pu donner leur avis sur celui-ci que les remarques deviendront réellement significatives par rapport au programme réalisé.

CHAPITRE 8
PROJETS D'AVENIR
ET DEVELOPPEMENTS FUTURS

Maintenant que notre programme est réalisé, implémenté, testé, il nous reste à envisager le futur. Dans ce chapitre, nous allons essayer d'établir une liste (non exhaustive bien entendu) des différentes améliorations et des ajouts qui augmenteraient la qualité et les possibilités du logiciel Cyrano.

8.1 Le programme réalisé

La rapidité d'expression

Au niveau de l'utilisation de notre programme, une augmentation générale de la rapidité d'expression entraînerait certainement plus de satisfactions de la part de l'utilisateur. La chose à prendre en compte pour réaliser cette augmentation de rapidité d'expression est bien entendu la vitesse d'exécution du programme.

Nous parlons ici des performances du moteur vocal. Les développeurs de *MBROLA* à Mons testent pour l'instant une librairie Windows (*DLL*) offrant les mêmes fonctionnalités que le programme. Grâce à cela, le délai écoulé entre la demande de prononciation par l'utilisateur et la prononciation effective pourra être réduit (l'exécution d'un programme prend un certain temps dû à la mise en place de l'environnement du programme).

De plus, il est prévu que cette librairie accède elle-même à la carte son pour demander la prononciation des fichiers sonores pendant leur génération. En utilisant cette méthode, un temps précieux serait gagné puisqu'il ne serait plus nécessaire d'attendre la génération complète du fichier sonore pour en entendre le contenu.

L'interface à pictogrammes

Dans l'état actuel des choses, l'interface à pictogrammes est utile pour les enfants ainsi que pour les personnes mentalement handicapées. Il serait intéressant de chercher à étendre son intérêt, vers les personnes physiquement handicapées par exemple.

En l'équipant d'un système de pointage à balayage, nous pourrions toucher un tel public. Dans ce cas, il serait même plus intéressant d'en faire un programme indépendant, vu la différence entre les utilisateurs ciblés par ce nouveau logiciel et notre Cyrano.

Les phrases préfabriquées

Le développement de l'interface à pictogrammes n'est pas la seule possibilité de transformation du programme. Il serait par exemple envisageable d'ajouter des caractéristiques rendant les phrases préfabriquées plus intéressantes.

Nous avions, notamment, envisagé de laisser la possibilité de donner à chaque phrase préfabriquée un raccourci (suite de caractères) permettant d'y faire appel en cours d'utilisation du programme (l'utilisateur tape au clavier le raccourci de la phrase plutôt que d'utiliser la souris pour demander la prononciation). Le temps d'accès n'en aurait été que plus court et les résultats donc meilleurs. Le temps dont nous disposions pour développer le programme ne nous permettant pas d'implémenter cette caractéristique, elle pourra être envisagée lors des développements futurs.

Pour augmenter la qualité des phrases proprement dites, on pourrait imaginer un petit atelier de création de phrases dans lequel l'utilisateur pourrait régler précisément la hauteur de ton sur chaque phonème (par exemple en présentant la phrase visuellement dans une fenêtre et en indiquant, à l'aide de curseurs répartis sur la longueur de la phrase, la hauteur de ton de chaque partie de celle-ci) et rendre ses phrases plus agréables à l'oreille.

8.2 Les ajouts possibles

Les langues

MBROLA offre de nombreuses bases de données dans des langues différentes. Il est donc possible de changer la langue d'expression du programme. Toutefois, la taille des bases de données est telle que la place occupée par les données du programme serait énorme s'il fallait proposer toutes les langues simultanément. C'est pourquoi l'utilisateur ne peut choisir nulle part la langue d'expression. Il paraît plus judicieux de créer plusieurs programmes proposant chacun une langue différente et de rendre les différentes versions disponibles indépendamment.

Le projet global

Si nous considérons le projet global de synthèse vocale pour tous les types de handicapés, il est clair que la tâche à accomplir est colossale. C'est pourquoi, dans le chapitre 2, il nous a semblé important de cibler précisément le public visé et de résoudre un seul type de problème parmi tous ceux existants.

Il serait fort peu intéressant de réaliser un programme répondant à tous les besoins en une fois. Par contre, Cyrano peut tout à fait être perçu comme première tentative de développement d'un logiciel dans l'ensemble des programmes utiles dans le domaine du mutisme. Nous avons déjà indiqué, au point 8.1, un autre type de programme réalisable dans ce cadre. De nombreuses autres possibilités existent. Elles

devront sans doute être imaginées en fonction des besoins qui peuvent se présenter et des problèmes qui peuvent survenir.

8.3 Autre dimension de la synthèse vocale

L'apport de l'électronique

Pour notre programme, l'idéal serait que l'utilisateur puisse ne plus s'en séparer. On n'est pas muet uniquement pendant les heures de bureau. Pour être réellement utile, le programme Cyrano devrait être disponible à chaque instant où l'utilisateur souhaite s'exprimer verbalement. Malheureusement, la technologie des ordinateurs actuels en rend le transport malaisé, même pour les PC portables, et le temps de mise en route est certainement beaucoup plus long que ce qui serait acceptable.

Une solution à ce problème serait d'utiliser un boîtier électronique capable d'effectuer la synthèse vocale, disposant d'une autonomie suffisante pour être disponible tout au long de la journée et de taille suffisamment réduite pour être transporté commodément.

Ce type de boîtier existe déjà à l'heure actuelle mais n'offre pas encore la possibilité de préfabriquer des phrases phonétiques. L'idée serait donc d'inventer un boîtier capable de communiquer avec notre programme. Dans ce cas, il serait alors possible à l'utilisateur de se servir de notre programme pour gérer sa structure de groupes puis il pourrait enregistrer les phrases préfabriquées dans le boîtier.

Nous sommes bien conscients qu'il s'agit là d'un défi lancé à l'électronique, toutefois, si cette coopération réussissait, ce serait un pas en avant considérable pour l'aide aux muets.

CONCLUSION

Maintenant que le programme est terminé et que ce mémoire va pouvoir être clôturé, peut-être serait-il intéressant de faire le point sur l'expérience que ce travail m'a permi d'acquérir. Je me permettrai d'être plus personnel dans cette partie de la rédaction pour vous expliquer ce que m'a apporté cette dernière réalisation en tant qu'étudiant. Je découperai mon propos en différents sujets que j'aborderai séparément.

Les contacts avec l'utilisateur ou le client

Le contact avec l'utilisateur est un sujet peu abordé au cours des études. Il a été étonnant pour moi de me trouver face à une personne qui ne savait pas exactement ce qu'elle désirait, ni ce qui était possible, mais qui souhaitait en tout cas que je réalise un programme pour elle.

La découverte de cette dimension de l'informatique m'a plutôt enchanté. Etant plus habitué à recevoir des spécifications détaillées qu'à essayer de découvrir ce qui est important et ce qui ne l'est pas pour l'utilisateur, je ne soupçonnais pas les difficultés qui m'attendaient.

Ecouter l'utilisateur. Comprendre ce qu'il veut lorsqu'il ne sait pas encore l'exprimer lui-même. Lui faire découvrir des possibilités inconnues sans orienter son choix. Rester neutre dans la décision de ce qui doit - ou ne doit pas - être réalisé. Autant de détails dont on ne peut ignorer l'importance. C'est ici que se joue le futur d'un programme.

Cette étape de la réalisation du programme fut d'autant plus enrichissante que je partais, moi-même, avec une idée préconçue du programme à réaliser. Sans avoir vraiment pris le temps de chercher à savoir ce qui pouvait bien être attendu par les utilisateurs de mon programme, j'avais déjà en tête tout un tas d'idées de ce que j'allais réaliser. Ce n'est que lorsque j'eus confronté mes idées avec celles des autres personnes impliquées dans le projet que je me rendis compte que le but n'était pas de vendre ses idées où de les faire admettre. Le plus important, il me semble, est de satisfaire l'utilisateur en lui offrant - non ce qu'il a demandé - mais bien ce qu'il souhaite. Ce n'est qu'au fil des discussions avec monsieur Lamy que j'ai fini par comprendre ce dont il avait besoin et donc le programme que je devais essayer de développer.

Je suis bien conscient que l'on arrive jamais totalement à comprendre les besoins ou les désirs de l'utilisateur, mais, en tout cas, je pense que seule la discussion et l'échange d'idées sur le logiciel à réaliser peuvent y parvenir.

La méthodologie et l'analyse

Comme nous l'avons vu dans le chapitre 3, mon souhait premier était d'appliquer la méthodologie *TRIDENT*. Comme nous l'avons vu également dans ce même chapitre, les résultats obtenus avec la méthodologie ne furent pas suffisament satisfaisants que pour être exploités.

A vrai dire, j'ai été fort déçu par cet échec méthodologique. J'attendais beaucoup d'une méthode capable de guider le développement à bon port. Force me fut de constater que l'application mécanique d'une méthode ne donne que des résultats mécaniques. De plus, il est possible que je n'aie pas appliqué correctement la méthodologie choisie. Si les méthodologies permettent d'introduire la rigueur dans la conception d'une application (surtout si cette application est de grande taille), elles ne semblent pas pouvoir aider l'inventivité ni contribuer à l'originalité.

Or, puisqu'il existe des cas où les méthodologies donnent d'excellents résultats, il faut supposer que notre projet ne s'inscrivait pas dans le cadre des emplois justifiés de la méthodologie *TRIDENT*. Peut-être existe-t-il une méthodologie qui aurait été plus adéquate pour Cyrano; étant donné que nous ne l'avons pas trouvée, nous avons dû nous rabattre vers un mode de développement moins systématique.

Les chapitres 4, 5 et 6 nous ont montré les difficultés qui sont apparues en employant ce mode de développement. Faute de lignes directrices précises, les différentes phases d'analyse et la programmation ont parfois été un peu mélées. Nous sommes parvenus à éviter les pièges de la mécanique méthodologique même si nous avons dû le payer en termes de netteté du travail.

La programmation

La programmation du logiciel reste pour moi l'étape qui m'a le plus appris sur mes capacités. En effet, la programmation Orientée Objet était restée, pour moi, jusqu'à ce mémoire, un concept un peu mystérieux.

Durant tout le travail de programmation, j'ai dû mettre en pratique très concrètement les choses apprises dans les cours de théorie des languages. J'ai également dû faire attention à mes réflexes de programmation procédurale. Le lecteur aura remarqué qu'à de nombreux endroits dans mon programme, les réflexes l'ont emporté... Dans la plupart des cas, il a été possible de corriger ces erreurs par une approche orientée objet correcte. Toutefois, pour certaines erreurs, modifier le programme aurait eut un impact trop important pour que cela soit fait.

L'utilisation des ressources, et, notamment, la gestion de celles-ci dans la modularité m'a également causé quelques difficultés.

Dans l'ensemble, la programmation ne fut pas la partie la plus délicate de ce travail (sans doute en raison de la pratique acquise au cours de mes années d'étude). Seul le manque de connaissances m'a parfois amené à tourner un peu en rond jusqu'à ce

que le professeur Cherton m'indique l'élément que je ne connaissait pas et qui m'était nécessaire.

D'une manière générale, j'y ai certainement appris à utiliser les fichiers d'aide et les livres de référence souvent trop délaissés.

La gestion de la charge de travail

Au départ, lorsque nous avons rencontré monsieur Lamy pour la première fois, je n'imaginais pas vraiment la quantité de travail qui m'attendait. A la limite, j'étais prêt à en faire toujours plus (pour la satisfaction de l'utilisateur) en pensant que cela ne prendrait somme toute pas trop de temps.

Lorsque j'ai été bien avancé dans la réalisation du programme, mais que je me suis rendu compte que le temps commençait à manquer, je me suis aperçu que le professeur Cherton avait eu bien raison de modérer mon ardeur. Sans cela, il est certain que j'aurais été amené à bâcler une partie du travail, faute de temps.

A présent, je me rend compte qu'il a sans doute été judicieux de voir petit (c'est-à-dire de prévoir de pouvoir réaliser peu de choses) tout en ayant ainsi pu apporter plus d'attention à l'ensemble du travail.

Toujours est-il que les prévisions n'avaient pas été si mauvaises puisque le logiciel est maintenant terminé et que je parviendrai sans doute à rendre ce mémoire dans les délais (...).

La collaboration

Dans ce travail, rien n'aurait été possible sans la collaboration dont j'ai bénéficié. Tout au long des chapitres et des lignes de code, le professeur Cherton se trouvait là pour réorienter le travail, attirer mon attention sur les points délicats ou litigieux. A de nombreuses reprises, nous avons été amenés à réfléchir sur le programme à réaliser et sur la manière de le réaliser. J'ai pu me rendre compte d'un certain nombre de choses concernant la collaboration au sein d'un projet.

Tout d'abord, il est difficile de parler des mêmes choses en même temps. On a parfois l'impression que la communication s'établi bien puis, plus tard, on se rend compte qu'en fait on ne parlait pas des mêmes choses.

De la même manière, lorsqu'une des deux personnes pense avoir compris le point de vue de l'autre, est-ce vraiment certain ? N'y a-t-il pas l'un ou l'autre détail qui fait que les représentations mentales qu'ont les deux personnes sont différentes ? Dans la plupart des cas, cela n'a pas d'importance. Mais pour certains points délicats, les conséquences peuvent parfois être étonnantes.

Je peux donc dire que j'ai beaucoup appris de cet échange d'idées qui a eu lieu tout au long de la réalisation du programme et de la rédaction du mémoire. Appris sur

la difficulté d'exprimer clairement son point de vue de manière telle que l'autre le comprenne sans ambiguïté. Appris également sur la difficulté de comprendre exactement ce qu'une personne souhaite exprimer, tout cela parce qu'elle n'exprime pas ses idées comme nous l'aurions fait. Appris enfin sur la difficulté de prendre des décisions avec une autre personne ou d'être réellement d'accord avec cette personne puisqu'on ne peut jamais savoir si on a vraiment bien compris son point de vue.

Bref...

Bref, j'ai beaucoup appris. J'ai également pu mettre en pratique un grand nombre de choses apprises au cours de mes études (programmation orientée objet, communication...).

Ce mémoire restera, à n'en pas douter, l'une des expériences les plus enrichissantes de mes années d'étude.

BIBLIOGRAPHIE

La méthode PHONEPIC

- *Les Téléthèses de Communication : l'Apport des Sciences du Langage à HECTOR,*
  in TRANEL (numéro spécial),
  Institut de Linguistique,
  Université de Neuchâtel, Septembre 1987.

- R. Jeanneret, F. Redard Abu-Rub,
  Une Entrée phonologique par Pictogrammes pour les Synthétiseurs de Parole à l'Usage des Handicapés moteurs cérébraux,
  in Bulletin CILA (numéro spécial),
  Centre de linguistique appliquée,
  Université de Neuchâtel, Mars 1989.

- F. Redard Abu-Rub,
  Utilisation de l'Ecriture phonétique selon le Système PHONEPIC,
  Centre de linguistique appliquée,
  Université de Neuchâtel, Juillet 1989.

- A. Matthey, F. GrosJean,
  L'Apport potentiel de l'Intelligence artificielle et du Traitement automatique du Langage naturel à une nouvelle Version d'HECTOR,
  in TRANEL (numéro spécial),
  Institut de Linguistique,
  Université de Neuchâtel, Septembre 1992.

La Méthodologie TRIDENT

- J. Vanderdonckt, F. Bodart,
  Encapsulating Knowledge for Intelligent Automatic Interaction Object Selection,
  Institut d'Informatique,
  Facultés Universitaires Notre-Dame de la Paix, Avril 1993.

- F. Bodart, A.-M. Hennebert, J.-M. Leheureux, I. Provot, J. Vanderdonckt, G. Zucchinetti,
  Key Activities for A Development Methodology of Interactive Applications,
  in *Critical Issues in User Interface Systems Engineering,*
  D.Benyon and Ph. Palanque (Eds.), Springer-Verlag, Berlin, 1995.

La phonétique

- B. Malmberg,
  La Phonétique,
  Que sais-je ?, Mars 1975.

- J.-L. Duchet,
  La Phonologie,
  Que sais-je ?, Novembre 1986.

- *A Speech Timing Model (1st state)*,
  Laboratoire d'Analyse Informatique de la Parole,
  University of Lausanne, Juillet 1993.

Le projet MBROLA

- T. Dutoit, V. Pagel, N. Pierret, F. Bataille, O. Van Der Vrecken,
  The MBROLA Project : Towards a Set of High-Quality Speech Synthesizers Free of Use for Non-Commercial Purposes,
  Philadelphia, 1996.

- T. Dutoit,
  An Introduction to Text-To-Speech Synthesis,
  Kluwer Academic Publishers, 1997.

ANNEXES

Annexe 1 - Les durées moyennes des phonèmes de la langue française

Ces durées ont été extraites de *BDTons* et du *corpus Ivan Levai*. Elles nous ont été fournies par les développeurs de *MBROLA* (Faculté Polytechnique de Mons)

Phonème (représentation utilisée dans Cyrano)	Durée (source 1) en millisecondes	Durée (source 2) en millisecondes
i	94	79
é	97	86
è	76	82
a	83	83
O	58	95
o	114	83
ù	77	87
u	91	74
e	70	75
î	108	96
â	131	111
ô	129	121
û	108	102
y	77	61
w	77	65
H	70	58
p	64	97
t	66	88
k	70	80
b	64	75
d	58	68
g	72	55
f	88	123
s	106	123
h	134	120
v	56	78
z	76	87
j	88	80
l	64	50
r	69	54
m	85	77
n	78	64
N	121	72

Annexe 2 - Le programme principal

```
{*************************}
{**  Fichier Cyrano.pas  **}
{*************************}

Program CYRANO;{Programme principal}


{Synthétiseur vocal concu et réalisé par Geoffroy Gailly}


{$R Cyrano.res}


Uses Objects, OWindows, WinDos, WinProcs, WinTypes,
     VarGlob, GestPar, GestGrp, CyrDlg;


Type TCyranoApp = Object (TApplication){Application de base}

                   Constructor Init;
                   Procedure   InitMainWindow; virtual;
                   Procedure   InitInstance; virtual;
                   Destructor  Done; virtual;

                  end;


Var CyranoApp:  TCyranoApp;{Instance de l'application}


{*****************************************************************}


Constructor TCyranoApp.Init;
{Construit l'application}

 begin

  CurseurFleche:= LoadCursor (HInstance, 'idc_Arrow');
  CurseurHP:= LoadCursor (HInstance, 'HautParleur');

  RegisterType (RElement);
  RegisterType (RReference);
  RegisterType (RGroupe);
  RegisterType (RCollection);
  StructGrp:= New (PStruct, Init);

  RegisterType (RParam);
  Param:= New (PParam, Init);

  TApplication.Init (NomAppl)

 end;


Procedure TCyranoApp.InitMainWindow;
{Initialise la fenêtre principale}

 var S: PCyrDlg;
     P: PCyrDlgPicto;
```

```
    G: PCyrDlgGroupes;

 begin

  S:= New (PCyrDlg);
  P:= New (PCyrDlgPicto);
  G:= New (PCyrDlgGroupes);

  S^.Init (Nil, Simple, S, P, G);
  P^.Init (Nil, S, P, G);
  G^.Init (Nil, S, P, G);

  case Param^.TypeInterf of
   Simple: begin
            P^.Create;
            G^.Create;
            MainWindow:= S
           end;
   Picto: begin
           S^.Create;
           G^.Create;
           MainWindow:= P
          end;
   Groupes: begin
             S^.Create;
             P^.Create;
             MainWindow:= G
            end
  end

 end;


Procedure TCyranoApp.InitInstance;
{Initialise l'instance courante de l'application}

 var D: PCyrDlg;

 begin

  TApplication.InitInstance;
  D:= PCyrDlg (MainWindow);
  HAccTable:= D^.Accel (HInstance)

 end;


Destructor TCyranoApp.Done;
{Détruit l'application}

 var finfo1: TSearchRec;
     finfo2: TOFStruct;

 begin

  DestroyCursor (CurseurHP);

  Dispose (Param, Done);{Suppression des paramètres}
  Dispose (StructGrp, Done);{Suppression de la structure de
groupe}

  FindFirst ('*.WAV', faAnyFile, finfo1);
  while DosError = 0 do begin
   OpenFile (finfo1.Name, finfo2, of_Delete);
```

```
   FindNext (finfo1)
  end;
  OpenFile ('phrase.pho', finfo2, of_Delete);

  TApplication.Done;

  UpdateWindow (Ecran);

 end;


{************************************}
{***   Corps du Programme CYRANO   ***}
{************************************}


Begin

 Ecran:= GetFocus;

 CyranoApp.Init;
 CyranoApp.Run;
 CyranoApp.Done

End.
```

Annexe 3 - La *unit CyrDlg*

```
{*************************}
{**  Fichier cyrdlg.pas  **}
{*************************}


Unit CyrDlg;{Fenêtre principale de l'application}


{$R CyrDlg.res}


Interface


Uses ODialogs, OWindows, WinTypes,
     CyrTyp, GestPar;


Const {Pour CyrDlg}
      cm_Prononcer =  101;
      cm_Parametres = 102;
      cm_Resume =     103;
      cm_Quitter =    104;

      cm_Groupes =     105;
      cm_Phrases =     106;
      cm_FairePhrase = 107;

      cm_Index = 108;
      cm_API =   109;
      cm_APD =   110;

      id_etTexte =     111;
      id_lbAncTexte =  112;
      id_pbPrononcer = 113;
      id_sbVolume =    114;
      id_pbAide =      116;
      id_stAide =      117;

      {Pour TCyrDlgPicto}
      id_pbf = 130; id_pbs = 131; id_pbch = 132; id_pbu = 133;
id_pbou = 134;
      id_pbv = 135; id_pbz = 136; id_pbj = 137; id_pbi = 138;
id_pbo = 139;
      id_pbp = 140; id_pbt = 141; id_pbk = 142; id_pby = 143;
id_pbeu = 144; id_pban = 145;
      id_pbb = 146; id_pbd = 147; id_pbg = 148; id_pbr = 149;
id_pbet = 150; id_pbon = 151;
      id_pbm = 152; id_pbn = 153; id_pbl = 154; id_pb_ = 155;
id_pba = 156; id_pbun = 157;
      id_pbeff = 162; id_pbaff = 159; id_pbexcl = 160; id_pbquest
= 161;

      {Pour TCyrDlgGroupes}
      id_stGroupeG =  164;
      id_lbGroupesG = 165;
      id_lbPhrasesG = 166;
      id_stGroupeD =  167;
      id_lbGroupesD = 168;
      id_lbPhrasesD = 169;
```

```
      id_lbDernUt =   170;
      id_lbPlusUt =   171;


Type PCyrPhonButton = ^TCyrPhonButton;
     TCyrPhonButton = Object (TCyrButton)

                       car: PChar;

                       Constructor InitResource (AParent: PWin-
dowsObject; ResourceID: Word; St: PCyrStatic; Txt: String; Caract:
PChar);
                       Procedure wmLButtonUp (var Msg: TMessage);
                                 virtual wm_First+wm_LButtonUp;
                       Procedure wmRButtonUp (var Msg: TMessage);
                                 virtual wm_First+wm_RButtonUp;

                      end;


     PCyrDlgPicto = ^TCyrDlgPicto;


     PCyrListeGrp = ^TCyrListeGrp;
     TCyrListeGrp = Object (TCyrListBox)

                     Procedure wmMouseMove (var Msg: TMessage);
                               virtual wm_First+wm_MouseMove;
                     Procedure wmLButtonUp (var Msg: TMessage);
                               virtual wm_First+wm_LButtonUp;
                     Procedure wmRButtonUp (var Msg: TMessage);
                               virtual wm_First+wm_RButtonUp;

                    end;


     PCyrListePhr = ^TCyrListePhr;
     TCyrListePhr = Object (TCyrListBox)

                     Procedure wmMouseMove (var Msg: TMessage);
                               virtual wm_First+wm_MouseMove;
                     Procedure wmLButtonUp (var Msg: TMessage);
                               virtual wm_First+wm_LButtonUp;
                     Procedure wmRButtonUp (var Msg: TMessage);
                               virtual wm_First+wm_RButtonUp;

                    end;


     PCyrEditPhon = ^TCyrEditPhon;
     TCyrEditPhon = Object (TCyrEdit)

                     Procedure wmRButtonUp (var Msg: TMessage);
                               virtual wm_First+wm_RButtonUp;

                    end;


     PCyrDlgGroupes = ^TCyrDlgGroupes;


     PCyrDlg = ^TCyrDlg;
     TCyrDlg = Object (TDlgWindow)
```

```
                TypInt:     TInterf;
                BteSimple:  PCyrDlg;
                BtePicto:   PCyrDlgPicto;
                BteGroupes: PCyrDlgGroupes;

                stAide: PCyrStatic;
                etTexte: PCyrEditPhon;
                lbAncTexte: PCyrListePhr;
                pbPrononcer, pbAide: PCyrButton;
                sbVolume: PCyrScrollBar;

                Constructor Init (AParent: PWindowsObject; TI:
TInterf; S: PCyrDlg; P: PCyrDlgPicto; G: PCyrDlgGroupes);
                Function    GetClassName: PChar; virtual;
                Procedure   GetWindowClass (var WC: TWndClass);
                              virtual;
                Procedure   SetupWindow; virtual;
                Function    Accel (Inst: THandle): THandle;
                              virtual;
                Procedure   wmMouseMove (var Msg: TMessage);
                              virtual wm_First+wm_MouseMove;
                Destructor  Done; virtual;
                Destructor  Fin; virtual;

                Procedure MiseAJour; virtual;

                Procedure Prononcer;
                             virtual cm_First+cm_Prononcer;
                Procedure Parametres;
                             virtual cm_First+cm_Parametres;
                Procedure Resume; virtual cm_First+cm_Resume;
                Procedure Quitter; virtual cm_First+cm_Quitter;

                Procedure GestGroupes;
                             virtual cm_First+cm_Groupes;
                Procedure GestPhrases;
                             virtual cm_First+cm_Phrases;
                Procedure FairePhrase;
                             virtual cm_First+cm_FairePhrase;

                Procedure API; virtual cm_First+cm_API;
                Procedure Aide; virtual cm_First+cm_Index;
                Procedure AProposDe; virtual cm_First+cm_APD;

                Procedure RacPrononcer (var Msg: TMessage);
                          virtual id_First+id_pbPrononcer;
                Procedure RacVolume (var Msg: TMessage);
                          virtual id_First+id_sbVolume;
                Procedure RacAide (var Msg: TMessage);
                          virtual id_First+id_pbAide;

              end;


     TCyrDlgPicto = Object (TCyrDlg)

                    pbf, pbs, pbch, pbu, pbou, pbv, pbz, pbj,
pbi, pbo, pbp, pbt, pbk, pby, pbeu, pban, pbb, pbd, pbg, pbr,
pbet, pbon, pbm, pbn, pbl, pb_, pba, pbun, pbaff, pbexcl, pbquest:
PCyrPhonButton;
                    pbeff: PCyrButton;

                    Constructor Init (AParent: PWindowsObject; S:
PCyrDlg; P: PCyrDlgPicto; G: PCyrDlgGroupes);
```

```
                    Procedure Effacer (var Msg: TMessage);
                              virtual id_First+id_pbEff;

                  end;


     TCyrDlgGroupes = Object (TCyrDlg)

                       stGroupeG, stGroupeD: PCyrStatic;
                       lbGroupesG, lbGroupesD: PCyrListeGrp;
                       lbPhrasesG, lbPhrasesD: PCyrListePhr;
                       lbDernUt, lbPlusUt: PCyrListeGrp;

                       Constructor Init (AParent: PWindowsObject;
S: PCyrDlg; P: PCyrDlgPicto; G: PCyrDlgGroupes);
                       Procedure SetupWindow; virtual;

                       Procedure MAJGroupes; virtual;

                      end;


{*************************************************************}


Implementation


Uses Objects, WinProcs,
     VarGlob, MotVoc, GestGrp, StrServ;


{*************************************************************}
{TCyrPhonButton}


Constructor TCyrPhonButton.InitResource (AParent: PWindowsObject;
     ResourceID: Word; St: PCyrStatic; Txt: String; Caract: Char);

 begin

  TCyrButton.InitResource (AParent, ResourceID, St, Txt);
  Car:= Caract

 end;


Procedure TCyrPhonButton.wmLButtonUp (var Msg: TMessage);

 var dlgtmp: PCyrDlg;

 begin

  DefWndProc (Msg);
  dlgtmp:= PCyrDlg (Parent);
  dlgtmp^.etTexte^.Insert (Car);
  dlgtmp^.etTexte^.SetSelection (256, 256)

 end;


Procedure TCyrPhonButton.wmRButtonUp (var Msg: TMessage);
```

```
 var dlgtmp: PCyrDlg;
     anctxt: String;

 begin

  DefWndProc (Msg);
  dlgtmp:= PCyrDlg (Parent);
  dlgtmp^.etTexte^.LitStr (anctxt);
  dlgtmp^.etTexte^.SetText (Car);
  dlgtmp^.Prononcer;
  dlgtmp^.etTexte^.EcritStr (anctxt);
  dlgtmp^.etTexte^.SetSelection (256, 256)

 end;


{*************************************************************}
{TCyrDlgPicto}


Constructor TCyrDlgPicto.Init (AParent: PWindowsObject; S:
PCyrDlg; P: PCyrDlgPicto; G: PCyrDlgGroupes);

 begin

  TCyrDlg.Init (AParent, Picto, S, P, G);

  pbf:= New (PCyrPhonButton, InitResource (@self, id_pbf, stAide,
Str45, 'f'));
  pbs:= New (PCyrPhonButton, InitResource (@self, id_pbs, stAide,
Str46, 's'));
  pbch:= New (PCyrPhonButton, InitResource (@self, id_pbch,
stAide, Str47, 'h'));
  pbu:= New (PCyrPhonButton, InitResource (@self, id_pbu, stAide,
Str48, 'u'));
  pbou:= New (PCyrPhonButton, InitResource (@self, id_pbou,
stAide, Str49, 'ù'));
  pbv:= New (PCyrPhonButton, InitResource (@self, id_pbv, stAide,
Str50, 'v'));
  pbz:= New (PCyrPhonButton, InitResource (@self, id_pbz, stAide,
Str51, 'z'));
  pbj:= New (PCyrPhonButton, InitResource (@self, id_pbj, stAide,
Str52, 'j'));
  pbi:= New (PCyrPhonButton, InitResource (@self, id_pbi, stAide,
Str53, 'i'));
  pbo:= New (PCyrPhonButton, InitResource (@self, id_pbo, stAide,
Str54, 'o'));
  pbp:= New (PCyrPhonButton, InitResource (@self, id_pbp, stAide,
Str55, 'p'));
  pbt:= New (PCyrPhonButton, InitResource (@self, id_pbt, stAide,
Str56, 't'));
  pbk:= New (PCyrPhonButton, InitResource (@self, id_pbk, stAide,
Str57, 'k'));
  pby:= New (PCyrPhonButton, InitResource (@self, id_pby, stAide,
Str58, 'y'));
  pbeu:= New (PCyrPhonButton, InitResource (@self, id_pbeu,
stAide, Str59, 'e'));
  pban:= New (PCyrPhonButton, InitResource (@self, id_pban,
stAide, Str60, 'â'));
  pbb:= New (PCyrPhonButton, InitResource (@self, id_pbb, stAide,
Str61, 'b'));
  pbd:= New (PCyrPhonButton, InitResource (@self, id_pbd, stAide,
Str62, 'd'));
```

```
  pbg:= New (PCyrPhonButton, InitResource (@self, id_pbg, stAide,
Str63, 'g'));
  pbr:= New (PCyrPhonButton, InitResource (@self, id_pbr, stAide,
Str64, 'r'));
  pbet:= New (PCyrPhonButton, InitResource (@self, id_pbet,
stAide, Str65, 'é'));
  pbon:= New (PCyrPhonButton, InitResource (@self, id_pbon,
stAide, Str66, 'ô'));
  pbm:= New (PCyrPhonButton, InitResource (@self, id_pbm, stAide,
Str67, 'm'));
  pbn:= New (PCyrPhonButton, InitResource (@self, id_pbn, stAide,
Str68, 'n'));
  pbl:= New (PCyrPhonButton, InitResource (@self, id_pbl, stAide,
Str69, 'l'));
  pb_:= New (PCyrPhonButton, InitResource (@self, id_pb_, stAide,
Str70, ' '));
  pba:= New (PCyrPhonButton, InitResource (@self, id_pba, stAide,
Str71, 'a'));
  pbun:= New (PCyrPhonButton, InitResource (@self, id_pbun,
stAide, Str72, 'û'));
  pbeff:= New (PCyrButton, InitResource (@self, id_pbeff, stAide,
Str73));
  pbaff:= New (PCyrPhonButton, InitResource (@self, id_pbaff,
stAide, Str74, '.'));
  pbexcl:= New (PCyrPhonButton, InitResource (@self, id_pbexcl,
stAide, Str75, '!'));
  pbquest:= New (PCyrPhonButton, InitResource (@self, id_pbquest,
stAide, Str76, '?'))

 end;


Procedure TCyrDlgPicto.Effacer (var Msg: TMessage);

 var longueur: Integer;
     i:        Integer;
     texte:    Array [0..255] of Char;

 begin

  longueur:= etTexte^.GetTextLen;
  if longueur > 0 then begin
   for i:= 0 to 255 do texte[i]:= ' ';
   etTexte^.GetText (texte, longueur);
   etTexte^.SetText (texte)
  end;
  etTexte^.SetSelection (256, 256)

 end;


{*************************************************************}
{TCyrListeGrp}


Procedure TCyrListeGrp.wmMouseMove (var Msg: TMessage);

 var index:    Integer;
     pas:      Integer;
     ancindex: Integer;

 begin

  TCyrListBox.wmMouseMove (Msg);
```

```
  index:= SendMessage (hWindow, lb_GetTopIndex, 0, 0);
  pas:= (Msg.LParamHi div 14);
  index:= index + pas;
  ancindex:= GetSelIndex;
  if ancindex <> index then SetSelIndex (index)

 end;


Procedure TCyrListeGrp.wmLButtonUp (var Msg: TMessage);

 var strtmp: String;
     dlgtmp: PCyrDlg;
     grp:    PGroupe;
     i:      Integer;

 function EstGrp (g: PGroupe): Boolean; far;
  begin EstGrp:= g^.Nom = strtmp end;

 begin

  DefWndProc (Msg);
  LitStr (strtmp, GetSelIndex);
  if (GetSelIndex >= 0) and (strtmp <> '') then Param^.GrpG:=
strtmp;
  grp:= StructGrp^.FirstThat (@EstGrp);
  if grp <> nil then begin
   grp^.NbreAcces:= grp^.NbreAcces+1;
   dlgtmp:= PCyrDlg (Parent);
   for i:= 0 to dlgtmp^.BteGroupes^.lbDernUt^.GetCount-1 do begin
    dlgtmp^.BteGroupes^.lbDernUt^.LitStr (strtmp, i);
    if strtmp = grp^.Nom then
dlgtmp^.BteGroupes^.lbDernUt^.DeleteString (i)
   end;
   dlgtmp^.BteGroupes^.lbDernUt^.EcritStr (grp^.Nom, 0);
   dlgtmp^.MiseAJour
  end

 end;


Procedure TCyrListeGrp.wmRButtonUp (var Msg: TMessage);

 var strtmp: String;
     dlgtmp: PCyrDlg;
     grp:    PGroupe;
     i:      Integer;

 function EstGrp (g: PGroupe): Boolean; far;
  begin EstGrp:= g^.Nom = strtmp end;

 begin

  DefWndProc (Msg);
  LitStr (strtmp, GetSelIndex);
  if (GetSelIndex >= 0) and (strtmp <> '') then Param^.GrpD:=
strtmp;
  grp:= StructGrp^.FirstThat (@EstGrp);
  if grp <> nil then begin
   grp^.NbreAcces:= grp^.NbreAcces+1;
   dlgtmp:= PCyrDlg (Parent);
   for i:= 0 to dlgtmp^.BteGroupes^.lbDernUt^.GetCount-1 do begin
    dlgtmp^.BteGroupes^.lbDernUt^.LitStr (strtmp, i);
```

```
    if strtmp = grp^.Nom then
dlgtmp^.BteGroupes^.lbDernUt^.DeleteString (i)
   end;
   dlgtmp^.BteGroupes^.lbDernUt^.EcritStr (grp^.Nom, 0);
   dlgtmp^.MiseAJour
  end

 end;


{*************************************************************}
{TCyrListePhr}


Procedure TCyrListePhr.wmMouseMove (var Msg: TMessage);

 var index, pas, ancindex: Integer;

 begin

  TCyrListBox.wmMouseMove (Msg);
  index:= SendMessage (hWindow, lb_GetTopIndex, 0, 0);
  pas:= (Msg.LParamHi div 14);
  index:= index + pas;
  ancindex:= GetSelIndex;
  if ancindex <> index then SetSelIndex (index)

 end;


Procedure TCyrListePhr.wmLButtonUp (var Msg: TMessage);

 var dlgtmp: PCyrDlg;
     strtmp: String;

 begin

  DefWndProc (Msg);
  dlgtmp:= PCyrDlg (Parent);
  LitStr (strtmp, GetSelIndex);
  dlgtmp^.etTexte^.AjouteStr (strtmp);
  dlgtmp^.etTexte^.SetSelection (256, 256)

 end;


Procedure TCyrListePhr.wmRButtonUp (var Msg: TMessage);

 var dlgtmp: PCyrDlg;
     anctxt, strtmp: String;

 begin

  DefWndProc (Msg);
  dlgtmp:= PCyrDlg (Parent);
  dlgtmp^.etTexte^.LitStr (anctxt);
  LitStr (strtmp, GetSelIndex);
  dlgtmp^.etTexte^.EcritStr (strtmp);
  dlgtmp^.Prononcer;
  dlgtmp^.etTexte^.EcritStr (anctxt);
  dlgtmp^.etTexte^.SetSelection (256, 256)

 end;
```

```
{*************************************************************}
{TCyrDlgGroupes}

Constructor TCyrDlgGroupes.Init (AParent: PWindowsObject; S:
PCyrDlg; P: PCyrDlgPicto; G: PCyrDlgGroupes);

 begin

  TCyrDlg.Init (AParent, Groupes, S, P, G);
  stGroupeG:= New (PCyrStatic, InitResource (@self, id_stGroupeG,
35));
  stGroupeD:= New (PCyrStatic, InitResource (@self, id_stGroupeD,
35));
  lbGroupesG:= New (PCyrListeGrp, InitResource (@self,
id_lbGroupesG, stAide, Str77));
  lbGroupesD:= New (PCyrListeGrp, InitResource (@self,
id_lbGroupesD, stAide, Str77));
  lbPhrasesG:= New (PCyrListePhr, InitResource (@self,
id_lbPhrasesG, stAide, Str78));
  lbPhrasesD:= New (PCyrListePhr, InitResource (@self,
id_lbPhrasesD, stAide, Str78));
  lbDernUt:= New (PCyrListeGrp, InitResource (@self, id_lbDernUt,
stAide, Str77));
  lbPlusUt:= New (PCyrListeGrp, InitResource (@self, id_lbPlusUt,
stAide, Str77))

 end;


Procedure TCyrDlgGroupes.SetupWindow;

 begin

  TCyrDlg.SetupWindow;
  stGroupeG^.SetText ('');
  stGroupeD^.SetText ('');
  MiseAJour

 end;


Procedure TCyrDlgGroupes.MAJGroupes;

 var coltmp:   PCollection;
     plafond:  LongInt;
     plancher: LongInt;
     i:        LongInt;

 Procedure RempllbGrpG (e: PElement); far;
  begin lbGroupesG^.EcritStr (e^.Nom, -1) end;
 Procedure RempllbPhrG (e: PElement); far;
  begin lbPhrasesG^.EcritStr (e^.Nom, -1) end;
 Procedure RempllbGrpD (e: PElement); far;
  begin lbGroupesD^.EcritStr (e^.Nom, -1) end;
 Procedure RempllbPhrD (e: PElement); far;
  begin lbPhrasesD^.EcritStr (e^.Nom, -1) end;
 Procedure RechPlafond (g: PGroupe); far;
  begin if g^.NbreActes > plafond then plafond:= g^.NbreActes end;
 Procedure RechPlancher (g: PGroupe); far;
  begin if g^.NbreActes < plancher then plancher:= g^.NbreActes
end;
 Procedure AffichPlancher (g: PGroupe); far;
  begin if g^.NbreActes = plancher then lbPlusUt^.EcritStr
(g^.Nom, 0) end;
```

```
 begin

  lbGroupesG^.ClearList;
  lbPhrasesG^.ClearList;
  lbGroupesD^.ClearList;
  lbPhrasesD^.ClearList;
  lbPlusUt^.ClearList;
  stGroupeG^.EcritStr (Str79+Param^.GrpG);
  coltmp:= StructGrp^.ListeGrpLies (Param^.GrpG);
  coltmp^.ForEach (@RempllbGrpG);
  coltmp:= StructGrp^.ListePhrLies (Param^.GrpG);
  coltmp^.ForEach (@RempllbPhrG);
  stGroupeD^.EcritStr (Str79+Param^.GrpD);
  coltmp:= StructGrp^.ListeGrpLies (Param^.GrpD);
  coltmp^.ForEach (@RempllbGrpD);
  coltmp:= StructGrp^.ListePhrLies (Param^.GrpD);
  coltmp^.ForEach (@RempllbPhrD);
  coltmp:= StructGrp^.ListeGrp;
  plafond:= -maxLongInt;
  coltmp^.ForEach (@RechPlafond);
  plancher:= +maxLongInt;
  coltmp^.ForEach (@RechPlancher);
  coltmp^.ForEach (@AffichPlancher);
  i:= plancher+1;
  for plancher:= i to plafond do coltmp^.ForEach (@AffichPlancher)

 end;


{*************************************************************}
{TCyrEditPhon}


Procedure TCyrEditPhon.wmRButtonUp (var Msg: TMessage);

 var dlgtmp: PCyrDlg;

 begin

  DefWndProc (Msg);
  dlgtmp:= PCyrDlg (Parent);
  dlgtmp^.Prononcer

 end;


{*************************************************************}
{TCyrDlg}


Constructor TCyrDlg.Init (AParent: PWindowsObject; TI: TInterf; S:
PCyrDlg; P: PCyrDlgPicto; G: PCyrDlgGroupes);
{Initialise la boîte de dialogue}

 begin

  TypInt:= TI;
  case TypInt of
   Simple: TDlgWindow.Init (AParent, NomAppl);
   Picto: TDlgWindow.Init (AParent, 'Picto');
   Groupes: TDlgWindow.Init (AParent, 'Groupes')
  end;
```

```
  BteSimple:= S;
  BtePicto:= P;
  BteGroupes:= G;
  stAide:= New (PCyrStatic, InitResource (@self, id_stAide, 150));
  etTexte:= New (PCyrEditPhon, InitResource (@self, id_etTexte,
256, stAide, Str1));
  lbAncTexte:= New (PCyrListePhr, InitResource (@self,
id_lbAncTexte, stAide, Str2));
  pbPrononcer:= New (PCyrButton, InitResource (@self,
id_pbPrononcer, stAide, Str3));
  sbVolume:= New (PCyrScrollBar, InitResource (@self, id_sbVolume,
stAide, Str4));
  pbAide:= New (PCyrButton, InitResource (@self, id_pbAide,
stAide, Str5))

 end;


Function TCyrDlg.GetClassName: PChar;
{Fournit le nom de classe}

 begin

  GetClassName:= NomAppl

 end;


Procedure TCyrDlg.GetWindowClass (var WC: TWndClass);
{Définit la classe de l'application}

 begin

  TDlgWindow.GetWindowClass (WC);
  WC.HIcon:= LoadIcon (HInstance, NomAppl)

 end;


Procedure TCyrDlg.SetupWindow;
{Met la boîte en place}

 begin

  TDlgWindow.SetupWindow;
  etTexte^.SetText ('');
  lbAncTexte^.ClearList;
  sbVolume^.SetupWindow;
  stAide^.SetText ('');
  MiseAJour

 end;


Function TCyrDlg.Accel (Inst: THandle): THandle;
{Charge les accélérateurs propres à la fenêtre}

 begin

  Accel:= LoadAccelerators (Inst, NomAppl)

 end;
```

```
Procedure TCyrDlg.wmMouseMove (var Msg: TMessage);
{Réagit à un mouvement de la souris}

 begin

  DefWndProc (Msg);
  stAide^.EcritStr ('')

 end;


Destructor TCyrDlg.Done;
{Détruit proprement la fenêtre}

 begin

  case TypInt of
   Simple: begin
            BtePicto^.Fin;
            BteGroupes^.Fin;
            Fin
           end;
   Picto: begin
           BteSimple^.Fin;
           BteGroupes^.Fin;
           Fin
          end;
   Groupes: begin
             BteSimple^.Fin;
             BtePicto^.Fin;
             Fin
            end
  end

 end;


Destructor TCyrDlg.Fin;
{Détruit la fenêtre}

 begin

  TDlgWindow.Done

 end;


Procedure TCyrDlg.MiseAJour;

 begin

  case Param^.TypeInterf of
   Simple: begin
            BteSimple^.Show (sw_Show);
            BtePicto^.Show (sw_Hide);
            BteGroupes^.Show (sw_Hide);
            BteSimple^.sbVolume^.SetPosition (100-param^.Volume)
           end;
   Picto: begin
           BtePicto^.Show (sw_Show);
           BteSimple^.Show (sw_Hide);
           BteGroupes^.Show (sw_Hide);
           BtePicto^.sbVolume^.SetPosition (100-param^.Volume)
          end;
```

```
   Groupes: begin
             BteGroupes^.Show (sw_Show);
             BteSimple^.Show (sw_Hide);
             BtePicto^.Show (sw_Hide);
             BteGroupes^.sbVolume^.SetPosition (100-
param^.Volume);
             BteGroupes^.MAJGroupes
            end
  end;
  ReglerVol (Param^.Volume)

 end;


Procedure TCyrDlg.Prononcer;
{Prononce le texte contenu dans la zone d'édition}

 var phrase:   String;
     i:        Integer;
     j:        LongInt;
     prononce: Boolean;
     ancphr:   String;
     inttmp:   LongInt;
     fichnom:  String;

 begin

  stAide^.EcritStr (Str6);
  SetCursor (CurseurHP);
  etTexte^.LitStr (phrase);
  if phrase <> '' then begin
   i:= 0;
   prononce:= False;
   while (i <= lbAncTexte^.GetCount-1) and (prononce = False) do
begin
    lbAncTexte^.LitStr (ancphr, i);
    if phrase = ancphr then begin
     inttmp:= Param^.NumPremF+lbAncTexte^.GetCount-i-1;
     Str (inttmp, fichnom);
     fichnom:= fichnom+'.WAV';
     PrononcerWav (fichnom);
     prononce:= True
    end
    else i:= i+1
   end;
   if prononce = false then begin
    Str (Param^.NumFich, fichnom);
    fichnom:= fichnom+'.WAV';
    CreerWav (phrase, etTexte^.GetTextLen-1, Param^.typevoix,
fichnom);
    PrononcerWav (fichnom);
    BteSimple^.lbAncTexte^.EcritStr (phrase, 0);
    BtePicto^.lbAncTexte^.EcritStr (phrase, 0);
    BteGroupes^.lbAncTexte^.EcritStr (phrase, 0);
    Param^.NumFich:= Param^.NumFich+1
   end
  end;
  etTexte^.SetText ('');
  SetCursor (CurseurFleche);
  stAide^.SetText ('');
  SetFocus (hWindow)

 end;
```

```
Procedure TCyrDlg.Parametres;
{Demande la gestion des paramètres}

 begin

  Param^.Gest (@self);
  MiseAJour

 end;


Procedure TCyrDlg.Resume;
{Affiche dans le bloc-note les phrases prononcées au cours de la
session}

 var blocnote: Text;
     i:        Integer;
     ligne:    String;

 begin

  if lbAncTexte^.GetCount > 0 then begin
   Assign (blocnote, 'resume.txt');
   Rewrite (blocnote);
   for i:= lbAncTexte^.GetCount-1 downto 0 do begin
    lbAncTexte^.LitStr (ligne, i);
    writeln (blocnote, ligne)
   end;
   close (blocnote);
   WinExec ('notepad resume.txt', sw_show)
  end

 end;



Procedure TCyrDlg.Quitter;
{Quitte le programme}

 begin

  Done

 end;


Procedure TCyrDlg.GestGroupes;
{Demande la gestion des groupes de la structure de groupe}

 begin

  StructGrp^.Gest (@self, Groupe);
  BteGroupes^.MAJGroupes

 end;


Procedure TCyrDlg.GestPhrases;
{Demande la gestion des phrases de la structure de groupe}

 begin

  StructGrp^.Gest (@self, Phrase);
```

```
  BteGroupes^.MAJGroupes

 end;


Procedure TCyrDlg.FairePhrase;
{Demande la transformation de la phrase éditée en phrase préfabri-
quée}

 var phr:      String;
     grpprinc: PGroupe;

 begin

  etTexte^.LitStr (phr);
  grpprinc:= StructGrp^.Principal;
  if phr <> '' then begin
   if grpprinc <> nil then
    StructGrp^.AjoutPhr (phr, grpprinc^.Nom)
   else StructGrp^.AjoutPhr (phr, '');
   BteGroupes^.MAJGroupes;
  end

 end;


Procedure TCyrDlg.API;
{Affiche la boîte de dialogue 'API'}

 var dlgtmp: PDialog;

 begin

  dlgtmp:= New (PDialog, Init (@self, 'API'));
  dlgtmp^.Execute

 end;


Procedure TCyrDlg.AProposDe;
{Affiche la boîte de dialogue 'A Propos De'}

 var dlgtmp: PDialog;

 begin

  dlgtmp:= New (PDialog, Init (@self, 'A_Propos_De'));
  dlgtmp^.Execute

 end;


Procedure TCyrDlg.RacPrononcer (var Msg: TMessage);
{Raccourci vers la procedure TCyranoDlg.Prononcer}

 begin

  DefWndProc (Msg);

  Prononcer;
  SetFocus (hWindow)

 end;
```

```
Procedure TCyrDlg.RacVolume (var Msg: TMessage);
{Raccourci de règlage du volume}

 begin

  DefWndProc (Msg);

  Param^.Volume:= 100-sbVolume^.GetPosition;
  ReglerVol (Param^.Volume);
  SetFocus (hWindow);
  etTexte^.SetSelection (255, 255)

 end;


Procedure TCyrDlg.RacAide (var Msg: TMessage);
{Raccourci de demande d'aide}

 begin

  DefWndProc (Msg);
  API

 end;


End.
```

Annexe 4 - La *unit MotVoc*

```
{*************************}
{**  Fichier MotVoc.pas  **}
{*************************}


Unit MotVoc;{Moteur Vocal}


Interface

 Type TVoix = (Garcon, Fille, Homme, Femme);


 Procedure CreerWav (Txt: String; Long: Integer; Voix: TVoix;
NomFich: String);

 Function  PrononcerWav (NomFich: String): Boolean;

 Procedure ReglerVol (Volume: LongInt);


{*****************************************************************}


Implementation


Uses Strings, WinProcs, MMSystem;


Procedure CreerWav (Txt: String; Long: Integer; Voix: TVoix;
NomFich: String);
{Crée un fichier .WAV sur base des paramètres reçus}

 var texte:  Array [0..255] of Char;
     phrase: Text;
     i:      Integer;
     p0, v0: Integer;
     p1, v1: Integer;
     p2, v2: Integer;
     p3, v3: Integer;
     p4, v4: Integer;
     intonauto:    Boolean;
     intoninstant: Boolean;
     intonhaut:    Integer;
     codeerreur:   Integer;
     sx:       String;
     commande: String;
     pchtmp:   Array [0..255] of Char;
     fichwav:  Text;

 procedure EcrireSon (son: String; long: Integer; pos: Integer);

  var strtmp: String;

  begin
   Str (long, strtmp);
   Write (phrase, son, ' ', strtmp);
```

```
   if intonauto = True then begin
    if pos = p0 then Write (phrase, ' 1 ', v0);
    if pos = p1 then Write (phrase, ' 25 ', v1);
    if pos = p2 then Write (phrase, ' 50 ', v2);
    if pos = p3 then Write (phrase, ' 75 ', v3);
    if pos = p4 then Write (phrase, ' 99 ', v4)
   end
   else if intoninstant = True then begin
    Write (phrase, ' 50 ', intonhaut);
    intoninstant:= False
   end;
   Writeln (phrase)

  end;

 begin

  {Formatage du Texte}
  For i:= 0 to 255 do texte[i]:= ' ';
  StrPCopy (texte, Txt);
  {Création du fichier .pho}
  Assign (phrase, 'phrase.pho');
  Rewrite (phrase);
  intoninstant:= False;
  p0:= 1;
  p1:= (long-1) div 4;
  p2:= (long-1) div 2;
  p3:= 3*((long-1) div 4);
  p4:= long-1;
  if Texte[long] = '-' then intonauto:= False
  else intonauto:= True;
  if Texte[long] = '?' then begin
   v0:= 120; v1:= 175; v2:= 100; v3:= 140; v4:= 220
  end
  else if Texte[long] = '!' then begin
   v0:= 120; v1:= 175; v2:= 100; v3:= 140; v4:= 160
  end
  else begin v0:= 120; v1:= 175; v2:= 100; v3:= 140; v4:= 70 end;
  if voix = femme then
   begin
    v0:= (v0*3)div 5; v1:= (v1*3)div 5; v2:= (v2*3)div 5;
    v3:= (v3*3)div 5; v4:= (v4*3)div 5
   end;
  EcrireSon ('_', 100, 0);
  for i:= 0 to long do
   case Texte[i] of
    '-': intonauto:= False;
    '+': intonauto:= True;
    '0'..'9': begin intonauto:= False;
               intoninstant:= True;
               Val (Texte[i], intonhaut, codeerreur);
               intonhaut:= intonhaut*30;
               if voix = femme then intonhaut:= intonhaut*2
              end;
    '?': begin intonauto:= True;
               p0:= i+1; p1:= ((long-(i+1)) div 4)+i+1;
               p2:= ((long-(i+1)) div 2)+i+1;
               p3:= (3*((long-(i+1)) div 4))+i+1; p4:= long;
               v0:= 120; v1:= 175; v2:= 100; v3:= 140; v4:= 220
         end;
    '!': begin intonauto:= True;
               p0:= i+1; p1:= ((long-(i+1)) div 4)+i+1;
               p2:= ((long-(i+1)) div 2)+i+1;
               p3:= (3*((long-(i+1)) div 4))+i+1; p4:= long;
```

```
                v0:= 120; v1:= 175; v2:= 100; v3:= 140; v4:= 160
          end;
     '.': begin intonauto:= True;
                p0:= i+1; p1:= ((long-(i+1)) div 4)+i+1;
                p2:= ((long-(i+1)) div 2)+i+1;
                p3:= (3*((long-(i+1)) div 4))+i+1; p4:= long;
                v0:= 120; v1:= 175; v2:= 100; v3:= 140; v4:= 70
          end;
     'i': EcrireSon ('i', 94, i+1);
     'é': EcrireSon ('e', 97, i+1);
     'è': EcrireSon ('E', 76, i+1);
     'a': EcrireSon ('a', 83, i+1);
     'à': EcrireSon ('A', 83, i+1);
     'O': EcrireSon ('O', 58, i+1);
     'o': EcrireSon ('o', 114, i+1);
     'ù': EcrireSon ('u', 77, i+1);
     'u': EcrireSon ('y', 91, i+1);
     'e': EcrireSon ('@', 70, i+1);
     'î': EcrireSon ('e~', 108, i+1);
     'â': EcrireSon ('a~', 131, i+1);
     'ê': EcrireSon ('a~', 131, i+1);
     'ô': EcrireSon ('o~', 129, i+1);
     'û': EcrireSon ('9~', 108, i+1);
     'y': EcrireSon ('j', 77, i+1);
     'w': EcrireSon ('w', 77, i+1);
     'H': EcrireSon ('H', 70, i+1);
     'p': EcrireSon ('p', 64, i+1);
     't': EcrireSon ('t', 66, i+1);
     'k': EcrireSon ('k', 70, i+1);
     'b': EcrireSon ('b', 64, i+1);
     'd': EcrireSon ('d', 58, i+1);
     'g': EcrireSon ('g', 72, i+1);
     'f': EcrireSon ('f', 88, i+1);
     's': EcrireSon ('s', 106, i+1);
     'h': EcrireSon ('S', 134, i+1);
     'v': EcrireSon ('v', 56, i+1);
     'z': EcrireSon ('z', 76, i+1);
     'j': EcrireSon ('Z', 88, i+1);
     'l': EcrireSon ('l', 64, i+1);
     'r': EcrireSon ('R', 69, i+1);
     'm': EcrireSon ('m', 85, i+1);
     'n': EcrireSon ('n', 78, i+1);
     'N': EcrireSon ('N', 121, i+1);
     ' ': EcrireSon ('_', 200, i+1)
    end;
  EcrireSon ('_', long+1, 100);
  Flush (phrase);
  Close (phrase);
  {Lancement de MBROLA}
  case voix of
   Garcon: sx:= 'ga';
   Fille: sx:= 'fi';
   Homme: sx:= 'ho';
   Femme: sx:= 'fe'
  end;
  commande:= 'mbrola fr'+sx+' phrase.pho '+NomFich;
  for i:= 0 to 255 do pchtmp[i]:= ' ';
  StrPCopy (pchtmp, commande);
  WinExec (pchtmp, 0)

 end;

Function PrononcerWav (NomFich: String): Boolean;
{Production sonore}
```

```
 var pchtmp: Array [0..12] of Char;
     i:      Integer;
     joue:   Boolean;

 begin

  for i:= 0 to 12 do pchtmp [i]:= ' ';
  StrPCopy (pchtmp, NomFich);
  PrononcerWav:= SndPlaySound (pchtmp, snd_sync or snd_NoDefault)

 end;


Procedure ReglerVol (Volume: LongInt);
{Règle le volume}

 begin

  Volume:= Volume*$28f028f;
  waveOutSetVolume (waveOutGetNumDevs-1, Volume)

 end;


End.
```

Annexe 5 - La *unit GestGrp*

```
{**************************}
{**  Fichier GestGrp.pas  **}
{**************************}

Unit GestGrp; {Gestion de la structure de groupes}

Interface

Uses Objects, OWindows;

Type TypeEl   = (Phrase, Groupe);

     PElement = ^TElement;
     TElement = Object (TObject)

                 Nom:       String;
                 TypEl:     TypeEl;
                 Existence: Boolean;{false -> effacement demandé}

                 Constructor Init (TypeE: TypeEl; NomE: String);
                 Constructor Load (var flx: TStream);
                 Procedure   Store (var flx: TStream); virtual;

                end;

     PReference = ^TReference;
     TReference = Object (TObject)

                   Nom: String;
                   Ref: PElement;

                   Constructor Init (PEl: PElement);
                   Constructor Load (var flx: TStream);
                   Procedure   Store (var flx: TStream); virtual;
                   Destructor  Done; virtual;

                  end;

     PGroupe = ^TGroupe;
     TGroupe = Object (TElement)

                Effacable: Boolean;
                Lies:      PCollection;{PReference}
                NbreAcces: Integer;

                Constructor Init (NomE: String);
                Constructor Load (var flx: TStream);
                Procedure   Store (var flx: TStream); virtual;

               end;

     PStruct = ^TStruct;
     TStruct = Object (TCollection)

                Constructor Init;
```

```
                Destructor Done; virtual;

                Function ListeGrp: PCollection;{PGroupe}
                Function ListeGrpLies (Grp: String):
                             PCollection;{PGroupe}
                Function ListeGrpNonLies (Grp: String):
                             PCollection;{PGroupe}
                Function ListeGrpEff: PCollection;{PGroupe}
                Function ListePhr: PCollection;{PElement}
                Function ListePhrLies (Grp: String):
                             PCollection;{PElement}
                Function ListePhrNonLies (Grp: String):
                             PCollection;{PElement}
                Function ListePhrEff: PCollection;{PElement}
                Function Principal: PGroupe;
                Procedure AjoutGrp(NomNouv:String; NomLie:String);
                Procedure AjoutPhr(NomNouv:String; NomLie:String);
                Procedure SupprEl (NomSuppr: String);
                Procedure RenEl (NouvNom: String; AncNom: String);
                Procedure RecupEl (NomRecup: String);
                Procedure LierEl (el, gp: String);
                Procedure DelierEl (el, gp: String);
                Procedure Gest (AParent: PWindowsObject; TypGest:
TypeEl);

              end;


Const RElement: TStreamRec = (
       ObjType: 1001;
       VmtLink: Ofs (TypeOf (TElement)^);
       Load: @TElement.Load;
       Store: @TElement.Store);

      RReference: TStreamRec = (
       ObjType: 1002;
       VmtLink: Ofs (TypeOf (TReference)^);
       Load: @TReference.Load;
       Store: @TReference.Store);

      RGroupe: TStreamRec = (
       ObjType: 1003;
       VmtLink: Ofs (TypeOf (TGroupe)^);
       Load: @TGroupe.Load;
       Store: @TGroupe.Store);

      RCollection: TStreamRec = (
       ObjType: 1004;
       VmtLink: Ofs (TypeOf (TCollection)^);
       Load: @TCollection.Load;
       Store: @TCollection.Store);


{*****************************************************************}


Implementation


{$R GestGrp.res}


Uses ODialogs, WinProcs, WinTypes, Strings, BWCC,
     VarGlob, CyrTyp, MotVoc, StrServ;
```

```
Const {TDlgGestGrp}
      id_cbGroupe = 173;
      id_pbGroupes = 174;
      id_pbPhrases = 175;
      id_brGroupes = 176;
      id_brPhrases = 177;
      id_stLies =    178;
      id_lbLies =    179;
      id_pbLier =    180;
      id_pbDelier =  181;
      id_stNonLies = 182;
      id_lbNonLies = 183;
      id_pbNouveauGrp =   184;
      id_pbNouvellePhr =  185;
      id_pbSupprimerGrp = 186;
      id_pbSupprimerPhr = 187;
      id_pbRenommerGrp =  188;
      id_pbRenommerPhr =  189;
      id_pbRecupererGrp = 190;
      id_pbRecupererPhr = 191;
      id_pbFermer = 192;
      id_pbAide =   193;
      id_stAide =   194;

      {TGestEl}
      id_stHaut = 197;
      id_etHaut = 198;
      id_stBas =  199;
      id_lbBas =  200;
      id_lbHaut = 204;
      id_pbAppliquer = 201;
      id_pbFermer2 =   202;
      id_pbAide2 =     203;


Type PCyrListeElem = ^TCyrListeElem;
     TCyrListeElem = Object (TCyrListBox)

                      ListeLies: Boolean;
                      Constructor InitResource (AParent: PWin-
dowsObject; ResourceID: Word; St: PCyrStatic; Txt: String; ListeL:
Boolean);
                      Procedure wmLButtonDblClk (var Msg: TMes-
sage); virtual wm_First+wm_LButtonDblClk;
                      Procedure wmRButtonUp (var Msg: TMessage);
                                virtual wm_First+wm_RButtonUp;

                     end;

     PDlgGestGrp = ^TDlgGestGrp;
     TDlgGestGrp = Object (TDlgWindow)

                    struct:    PStruct;
                    ElemGere:  TypeEl;
                    stAide, stLies, stNonLies: PCyrStatic;
                    cbGroupe: PCyrComboBox;
                    pbGroupes, pbPhrases: PCyrButton;
                    brGroupes, brPhrases: PBDivider;
                    lbLies, lbNonLies: PCyrListeElem;
                    pbLier, pbDelier, pbNouvGrp, pbNouvPhr,
pbSupprGrp, pbSupprPhr, pbRenGrp, pbRenPhr, pbRecupGrp,  pbRe-
cupPhr, pbFermer, pbAide: PCyrButton;
```

```
                    Constructor Init (AParent: PWindowsObject;
AGerer: TypeEl; Str: PStruct);
                    Procedure   SetupWindow; virtual;
                    Procedure   wmMouseMove (var Msg: TMessage);
                                virtual wm_First+wm_MouseMove;
                    Procedure MiseAJour; virtual;
                    Procedure ChgGrp (var Msg: TMessage);
                              virtual id_First+id_cbGroupe;
                    Procedure PoussGrp (var Msg: TMessage);
                              virtual id_First+id_pbGroupes;
                    Procedure PoussPhr (var Msg: TMessage);
                              virtual id_First+id_pbPhrases;
                    Procedure Lier (var Msg: TMessage);
                              virtual id_First+id_pbLier;
                    Procedure Delier (var Msg: TMessage);
                              virtual id_First+id_pbDelier;
                    Procedure NouvGrp (var Msg: TMessage);
                              virtual id_First+id_pbNouveauGrp;
                    Procedure NouvPhr (var Msg: TMessage);
                              virtual id_First+id_pbNouvellePhr;
                    Procedure SupprGrp (var Msg: TMessage);
                              virtual id_First+id_pbSupprimerGrp;
                    Procedure SupprPhr (var Msg: TMessage);
                              virtual id_First+id_pbSupprimerPhr;
                    Procedure RenGrp (var Msg: TMessage);
                              virtual id_First+id_pbRenommerGrp;
                    Procedure RenPhr (var Msg: TMessage);
                              virtual id_First+id_pbRenommerPhr;
                    Procedure RecupGrp (var Msg: TMessage);
                              virtual id_First+id_pbRecupererGrp;
                    Procedure RecupPhr (var Msg: TMessage);
                              virtual id_First+id_pbRecupererPhr;
                    Procedure Fermer (var Msg: TMessage);
                              virtual id_First+id_pbFermer;
                    Procedure Aide (var Msg: TMessage);
                              virtual id_First+id_pbAide;

                   end;


     PCyrListeGest = ^TCyrListeGest;
     TCyrListeGest = Object (TCyrListBox)

                      Procedure wmLButtonDblClk (var Msg: TMes-
sage); virtual wm_First+wm_LButtonDblClk;

                     end;


     TGest    = (Aj, Suppr, Ren, Recup);
     PDlgElem = ^TDlgElem;
     TDlgElem = Object (TDlgWindow)

                 TypeGest: TGest;
                 stHaut, stBas: PCyrStatic;
                 etHaut: PCyrEdit;
                 lbHaut, lbBas: PCyrListeGest;
                 pbAppliquer: PCyrButton;

                 Constructor Init (FenMere: PWindowsObject; TypeG:
TGest);
                 Procedure   SetupWindow; virtual;
                 Procedure   MiseAJour; virtual;
                 Procedure Appliquer (var Msg: TMessage);
```

```
                    virtual id_First+id_pbAppliquer;
                Procedure Fermer (var Msg: TMessage);
                    virtual id_First+id_pbFermer2;
                Procedure Aide (var Msg: TMessage);
                    virtual id_First+id_pbAide2;

              end;


{*************************************************************}
{TCyrListeElem}


Constructor TCyrListeElem.InitResource (AParent: PWindowsObject;
  ResourceID: Word; St: PCyrStatic; Txt: String; ListeL: Boolean);

 begin

  TCyrListBox.InitResource (AParent, ResourceID, St, Txt);
  ListeLies:= ListeL

 end;


Procedure TCyrListeElem.wmLButtonDblClk (var Msg: TMessage);

 var dlgtmp: PDlgGestGrp;

 begin

  DefWndProc (Msg);
  dlgtmp:= PDlgGestGrp (Parent);
  if ListeLies = True then dlgtmp^.Delier (Msg)
  else dlgtmp^.Lier (Msg)

 end;


Procedure TCyrListeElem.wmRButtonUp (var Msg: TMessage);

 var phr, fichnom: String;
     pos, long:    Integer;
     i:            LongInt;
     e:            PElement;
     pchtmp:       Array [0..255] of Char;

 Function EstPhr (p: PElement): Boolean; far;
  begin EstPhr:= p^.Nom = phr end;

 begin

  SetCursor (CurseurHP);
  DefWndProc (Msg);
  pos:= SendMessage (hWindow, lb_GetTopIndex, 0, 0);
  pos:= pos + (Msg.LParamHi div 14);
  SetSelIndex (pos);
  LitStr (phr, GetSelIndex);
  e:= StructGrp^.FirstThat (@EstPhr);
  if e <> nil then begin
   long:= GetString (pchtmp, GetSelIndex);
   Str (Param^.NumFichG, fichnom);
   fichnom:= 'G'+fichnom+'.WAV';
   CreerWav (phr, long, Param^.typevoix, fichnom);
   PrononcerWav (fichnom);
```

```
   Param^.NumFichG:= Param^.NumFichG+1;
   SetCursor (CurseurFleche)
  end

 end;


{*************************************************************}
{TDlgGestGrp}


Constructor TDlgGestGrp.Init (AParent: PWindowsObject; AGerer:
TypeEl; Str: PStruct);
{Construit la boîte de gestion de la structure de groupes}

 begin

  TDlgWindow.Init (AParent, 'Structure');
  struct:= Str;
  ElemGere:= AGerer;
  stAide:= New (PCyrStatic, InitResource (@self, id_stAide, 83));
  stLies:= New (PCyrStatic, InitResource (@self, id_stLies, 32));
  stNonLies:=New(PCyrStatic,InitResource(@self,id_stNonLies, 32));
  cbGroupe:= New (PCyrComboBox, InitResource (@self, id_cbGroupe,
255, stAide, Str7));
  pbGroupes:= New (PCyrButton, InitResource (@self, id_pbGroupes,
stAide, Str8));
  pbPhrases:= New (PCyrButton, InitResource (@self, id_pbPhrases,
stAide, Str9));
  brGroupes:= New (PBDivider, InitResource (@self, id_brGroupes));
  brPhrases:= New (PBDivider, InitResource (@self, id_brPhrases));
  lbLies:= New (PCyrListeElem, InitResource (@self, id_lbLies,
stAide, Str10, True));
  lbNonLies:= New (PCyrListeElem, InitResource (@self,
id_lbNonLies, stAide, Str11, False));
  pbLier:= New (PCyrButton, InitResource (@self, id_pbLier,
stAide, Str12));
  pbDelier:= New (PCyrButton, InitResource (@self, id_pbDelier,
stAide, Str13));
  pbNouvGrp:= New (PCyrButton, InitResource (@self,
id_pbNouveauGrp, stAide, Str14));
  pbNouvPhr:= New (PCyrButton, InitResource (@self,
id_pbNouvellePhr, stAide, Str15));
  pbSupprGrp:= New (PCyrButton, InitResource (@self,
id_pbSupprimerGrp, stAide, Str16));
  pbSupprPhr:= New (PCyrButton, InitResource (@self,
id_pbSupprimerPhr, stAide, Str17));
  pbRenGrp:= New (PCyrButton, InitResource (@self,
id_pbRenommerGrp, stAide, Str18));
  pbRenPhr:= New (PCyrButton, InitResource (@self,
id_pbRenommerPhr, stAide, Str19));
  pbRecupGrp:= New (PCyrButton, InitResource (@self,
id_pbRecupererGrp, stAide, Str20));
  pbRecupPhr:= New (PCyrButton, InitResource (@self,
id_pbRecupererPhr, stAide, Str21));
  pbFermer:= New (PCyrButton, InitResource (@self, id_pbFermer,
stAide, Str22));
  pbAide:= New (PCyrButton, InitResource (@self, id_pbAide,
stAide, Str5))

 end;


Procedure TDlgGestGrp.SetupWindow;
```

```
{Met en place la boîte de dialogue}

 begin

  TDlgWindow.SetupWindow;
  MiseAJour

 end;


Procedure TDlgGestGrp.wmMouseMove (var Msg: TMessage);
{Réagit à un mouvement de la souris dans la fenêtre}

 begin

  DefWndProc (Msg);
  stAide^.SetText ('')

 end;


Procedure TDlgGestGrp.MiseAJour;
{Met à jour les valeurs de la fenêtre}

 var coltmp: PCollection;
     strtmp: String;

 Procedure InsererGroupe (grp: PGroupe); far;
  begin cbGroupe^.EcritStr (grp^.Nom, -1) end;
 Procedure InsererLies (el: PElement); far;
  begin lbLies^.EcritStr (el^.Nom, -1) end;
 Procedure InsererNonLies (el: PElement); far;
  begin
   if el^.Nom<>Param^.GrpGere then lbNonLies^.EcritStr(el^.Nom,-1)
  end;

 begin

  cbGroupe^.ClearList;
  lbLies^.ClearList;
  lbNonLies^.ClearList;
  coltmp:= Struct^.ListeGrp;
  coltmp^.ForEach (@InsererGroupe);
  cbGroupe^.SelectStr (Param^.GrpGere);
  if cbGroupe^.GetSelIndex < 0 then begin
   cbGroupe^.SetSelIndex (0);
   cbGroupe^.LitStr (strtmp, cbGroupe^.GetSelIndex);
   Param^.GrpGere:= strtmp
  end;
  if ElemGere = Groupe then begin
   brGroupes^.Show (sw_Hide);
   brPhrases^.Show (sw_Show);
   stLies^.EcritStr (Str23);
   stNonLies^.EcritStr (Str24);
   coltmp:= Struct^.ListeGrpLies (Param^.GrpGere);
   coltmp^.ForEach (@InsererLies);
   coltmp:= Struct^.ListeGrpNonLies (Param^.GrpGere);
   coltmp^.ForEach (@InsererNonLies);
   pbNouvPhr^.Show (sw_Hide);
   pbSupprPhr^.Show (sw_Hide);
   pbRenPhr^.Show (sw_Hide);
   pbRecupPhr^.Show (sw_Hide);
   pbNouvGrp^.Show (sw_Show);
   pbSupprGrp^.Show (sw_Show);
```

```
   pbRenGrp^.Show (sw_Show);
   pbRecupGrp^.Show (sw_Show)
  end
  else begin
   brPhrases^.Show (sw_Hide);
   brGroupes^.Show (sw_Show);
   stLies^.EcritStr (Str25);
   stNonLies^.EcritStr (Str26);
   coltmp:= Struct^.ListePhrLies (Param^.GrpGere);
   coltmp^.ForEach (@InsererLies);
   coltmp:= Struct^.ListePhrNonLies (Param^.GrpGere);
   coltmp^.ForEach (@InsererNonLies);
   pbNouvGrp^.Show (sw_Hide);
   pbSupprGrp^.Show (sw_Hide);
   pbRenGrp^.Show (sw_Hide);
   pbRecupGrp^.Show (sw_Hide);
   pbNouvPhr^.Show (sw_Show);
   pbSupprPhr^.Show (sw_Show);
   pbRenPhr^.Show (sw_Show);
   pbRecupPhr^.Show (sw_Show)
  end

 end;


Procedure TDlgGestGrp.ChgGrp (var Msg: TMessage);
{Réagit à un changement de groupe}

 var grpsel: String;

 begin

  cbGroupe^.LitStr (grpsel, cbGroupe^.GetSelIndex);
  Param^.GrpGere:= grpsel;
  MiseAJour

 end;


Procedure TDlgGestGrp.PoussGrp (var Msg: TMessage);
{Réagit à un changement de type d'élément géré}

 begin

  ElemGere:= Groupe; MiseAJour

 end;


Procedure TDlgGestGrp.PoussPhr (var Msg: TMessage);
{Réagit à un changement de type d'élément géré}

 begin
  ElemGere:= Phrase; MiseAJour

 end;


Procedure TDlgGestGrp.Lier (var Msg: TMessage);
{Lie un élément au groupe géré}

 var element: String;

 begin
```

```
  lbNonLies^.LitStr (element, lbNonLies^.GetSelIndex);
  Struct^.LierEl (element, param^.GrpGere);
  MiseAJour

 end;


Procedure TDlgGestGrp.Delier (var Msg: TMessage);
{Délie un élément du groupe géré}

 var element: String;

 begin

  lbLies^.LitStr (element, lbLies^.GetSelIndex);
  Struct^.DelierEl (element, param^.GrpGere);
  MiseAJour

 end;


Procedure TDlgGestGrp.NouvGrp (var Msg: TMessage);
{Ouvre une boîte de création de groupe}

 var dlgtmp: PDlgElem;

 begin

  dlgtmp:= New (PDlgElem, Init (@self, Aj));
  dlgtmp^.Execute;
  MiseAJour

 end;


Procedure TDlgGestGrp.NouvPhr (var Msg: TMessage);
{Ouvre une boîte de création de phrase}

 var dlgtmp: PDlgElem;

 begin

  dlgtmp:= New (PDlgElem, Init (@self, Aj));
  dlgtmp^.Execute;
  MiseAJour

 end;


Procedure TDlgGestGrp.SupprGrp (var Msg: TMessage);
{Ouvre une boîte de suppression de groupe}
 var dlgtmp: PDlgElem;

 begin

  dlgtmp:= New (PDlgElem, Init (@self, Suppr));
  dlgtmp^.Execute;
  MiseAJour

 end;


Procedure TDlgGestGrp.SupprPhr (var Msg: TMessage);
{Ouvre une boîte de suppression de phrase}
```

```
 var dlgtmp: PDlgElem;

 begin

  dlgtmp:= New (PDlgElem, Init (@self, Suppr));
  dlgtmp^.Execute;
  MiseAJour

 end;


Procedure TDlgGestGrp.RenGrp (var Msg: TMessage);
{Ouvre une boîte de renommage de groupe}

 var dlgtmp: PDlgElem;

 begin

  dlgtmp:= New (PDlgElem, Init (@self, Ren));
  dlgtmp^.Execute;
  MiseAJour

 end;


Procedure TDlgGestGrp.RenPhr (var Msg: TMessage);
{Ouvre une boîte de modification de phrase}

 var dlgtmp: PDlgElem;

 begin

  dlgtmp:= New (PDlgElem, Init (@self, Ren));
  dlgtmp^.Execute;
  MiseAJour

 end;


Procedure TDlgGestGrp.RecupGrp (var Msg: TMessage);
{Ouvre une boîte de récupération de groupe}

 var dlgtmp: PDlgElem;

 begin

  dlgtmp:= New (PDlgElem, Init (@self, Recup));
  dlgtmp^.Execute;
  MiseAJour
 end;


Procedure TDlgGestGrp.RecupPhr (var Msg: TMessage);
{Ouvre une boîte de récupération de phrase}

 var dlgtmp: PDlgElem;

 begin

  dlgtmp:= New (PDlgElem, Init (@self, Recup));
  dlgtmp^.Execute;
  MiseAJour

 end;
```

```
Procedure TDlgGestGrp.Fermer (var Msg: TMessage);
{Clôture le dialogue}

 begin if CanClose then EndDlg (id_pbFermer) end;


{*************************************************************}
{TCyrListeGest}


Procedure TCyrListeGest.wmLButtonDblClk (var Msg: TMessage);

 var dlgtmp: PDlgElem;

 begin

  DefWndProc (Msg);
  dlgtmp:= PDlgElem (Parent);
  dlgtmp^.Appliquer (Msg)

 end;


{*************************************************************}
{TDlgElem}


Constructor TDlgElem.Init (FenMere: PWindowsObject; TypeG: TGest);
{Construit le dialogue}

 begin

  TDlgWindow.Init (FenMere, 'Gestion');
  TypeGest:= TypeG;
  stHaut:= New (PCyrStatic, InitResource (@self, id_stHaut, 58));
  etHaut:= New (PCyrEdit, InitResource (@self, id_etHaut, 255,
nil, ''));
  stBas:= New (PCyrStatic, InitResource (@self, id_stBas, 58));
  lbBas:=New(PCyrListeGest,InitResource(@self,id_lbBas, nil, ''));
  lbHaut:= New (PCyrListeGest, InitResource (@self, id_lbHaut,
nil, ''));
  pbAppliquer:= New (PCyrButton, InitResource (@self,
id_pbAppliquer, nil, ''))

 end;


Procedure TDlgElem.SetupWindow;
{Met le dialogue en place}

 begin

  TDlgWindow.SetupWindow;
  case TypeGest of
   Aj, Ren: begin lbHaut^.Show (sw_Hide);
                  etHaut^.Show (sw_Show);
                  stBas^.Show (sw_Show);
                  lbBas^.Show (sw_Show);
            end;
   Suppr, Recup: begin etHaut^.Show (sw_Hide);
                       stBas^.Show (sw_Hide);
                       lbBas^.Show (sw_Hide);
                       lbHaut^.Show (sw_Show);
                 end;
```

```
  end;
  MiseAJour

 end;


Procedure TDlgElem.MiseAJour;
{Met à jour le contenu de la boîte}

 var dlgtmp: PDlgGestGrp;
     coltmp: PCollection;

 Procedure RempllbBas (e: PElement); far;
  begin lbBas^.EcritStr (e^.Nom, -1) end;
 Procedure RempllbHaut (e: PElement); far;
  begin lbHaut^.EcritStr (e^.Nom, -1) end;

 begin

  dlgtmp:= PDlgGestGrp (Parent);
  case TypeGest of
   Aj: begin etHaut^.SetText ('');
             lbBas^.ClearList;
             coltmp:= dlgtmp^.struct^.ListeGrp;
             coltmp^.ForEach (@RempllbBas);
             lbBas^.EcritStr (Str27, 0);
             if dlgtmp^.ElemGere = Groupe then begin
              stHaut^.SetText (Str28);
              stBas^.SetText (Str29)
             end
             else begin
              stHaut^.SetText (Str30);
              stBas^.SetText (Str31)
             end;
             pbAppliquer^.SetCaption (Str32)
       end;
   Suppr: begin   lbHaut^.ClearList;
                  if dlgtmp^.ElemGere = Groupe then begin
                   stHaut^.SetText (Str33);
                   coltmp:= dlgtmp^.struct^.ListeGrp
                  end
                  else begin
                   stHaut^.SetText (Str34);
                   coltmp:= dlgtmp^.struct^.ListePhr
                  end;
                  coltmp^.ForEach (@RempllbHaut);
                  pbAppliquer^.SetCaption (Str35)
          end;
   Ren: begin     etHaut^.SetText ('');
                  lbBas^.ClearList;
                  if dlgtmp^.ElemGere = Groupe then begin
                   stHaut^.SetText (Str36);
                   stBas^.SetText (Str37);
                   coltmp:= dlgtmp^.struct^.ListeGrp;
                   pbAppliquer^.SetCaption (Str38)
                  end
                  else begin
                   stHaut^.SetText (Str39);
                   stBas^.SetText (Str40);
                   coltmp:= dlgtmp^.struct^.ListePhr;
                   pbAppliquer^.SetCaption (Str41)
                  end;
                  coltmp^.ForEach (@RempllbBas)
        end;
```

```
   Recup: begin   lbHaut^.ClearList;
                  if dlgtmp^.ElemGere = Groupe then begin
                   stHaut^.SetText (Str42);
                   coltmp:= dlgtmp^.struct^.ListeGrpEff
                  end
                  else begin
                   stHaut^.SetText (Str43);
                   coltmp:= dlgtmp^.struct^.ListePhrEff
                  end;
                  coltmp^.ForEach (@RempllbHaut);
                  pbAppliquer^.SetCaption (Str44)
          end
  end;

 end;


Procedure TDlgElem.Appliquer (var Msg: TMessage);
{Applique la gestion sélectionnée aux éléments sélectionnés}

 var dlgtmp:          PDlgGestGrp;
     strtmp, strtmp2: String;

 begin

  dlgtmp:= PDlgGestGrp (Parent);
  case TypeGest of
   Aj: begin
        etHaut^.LitStr (strtmp);
        if strtmp <> '' then begin
         lbBas^.LitStr (strtmp2, lbBas^.GetSelIndex);
         if dlgtmp^.ElemGere = Groupe then
          dlgtmp^.struct^.AjoutGrp (strtmp, strtmp2)
         else dlgtmp^.struct^.AjoutPhr (strtmp, strtmp2);
         MiseAJour
        end
       end;
   Suppr: begin
           lbHaut^.LitStr (strtmp, lbHaut^.GetSelIndex);
           dlgtmp^.struct^.SupprEl (strtmp);
           MiseAJour
          end;
   Ren: begin
         etHaut^.LitStr (strtmp);
         if strtmp <> '' then begin
          lbBas^.LitStr (strtmp2, lbBas^.GetSelIndex);
          dlgtmp^.struct^.RenEl (strtmp, strtmp2);
          MiseAJour
         end
        end;
   Recup: begin
           lbHaut^.LitStr (strtmp, lbHaut^.GetSelIndex);
           dlgtmp^.struct^.RecupEl (strtmp);
           MiseAJour
          end
  end

 end;


Procedure TDlgElem.Fermer (var Msg: TMessage);
{Clôture le dialogue}

 begin if canclose then EndDlg (id_pbFermer) end;
```

```
{*************************************************************}
{TElement}


Constructor TElement.Init (TypeE: TypeEl; NomE: String);
{Construit un nouvel élément}

 begin

  TObject.Init;
  Nom:= NomE;
  TypEl:= TypeE;
  Existence:= True

 end;


Constructor TElement.Load (var flx: TStream);
{Lit un élément dans un flux}

 var pchtmp: Array [0..255] of char;
     i:      Integer;
     car:    Char;

 begin

  for i:= 0 to 255 do pchtmp [i]:= ' ';
  flx.Read (pchtmp, SizeOf (pchtmp));
  Nom:= StrPas (pchtmp);
  flx.Read (car, SizeOf (car));
  if car = 'P' then typel:= Phrase
  else TypEl:= Groupe;
  Existence:= true
 end;


Procedure TElement.Store (var flx: TStream);
{Ecrit un élément dans un flux}

 var pchtmp: Array [0..255] of Char;
     i:      Integer;
     car:    Char;

 begin

  for i:= 0 to 255 do pchtmp [i]:= ' ';
  StrPCopy (pchtmp, Nom);
  flx.Write (pchtmp, SizeOf (pchtmp));
  if TypEl = Phrase then car:= 'P'
  else car:= 'G';
  flx.Write (car, SizeOf (car))

 end;


{*************************************************************}
{TReference}


Constructor TReference.Init (PEl: PElement);
{Initialise une nouvelle référence}

 begin
```

```
  TObject.Init;
  Nom:= PEl^.Nom;
  Ref:= PEl

 end;


Constructor TReference.Load (var flx: TStream);
{Lit une référence dans un flux}

 var pchtmp: Array [0..255] of char;
     i:      Integer;

 begin

  for i:= 0 to 255 do pchtmp [i]:= ' ';
  flx.Read (pchtmp, SizeOf (pchtmp));
  Nom:= StrPas (pchtmp);
  Ref:= nil

 end;


Procedure TReference.Store (var flx: TStream);
{Ecrit une référence dans un flux}

 var pchtmp: Array [0..255] of Char;
     i:      Integer;

 begin

  for i:= 0 to 255 do pchtmp [i]:= ' ';
  StrPCopy (pchtmp, Nom);
  flx.Write (pchtmp, SizeOf (pchtmp))

 end;

Destructor TReference.Done;
{Détruit une référence}

 begin

  Ref:= nil; TObject.Done

 end;


{*************************************************************}
{TGroupe}


Constructor TGroupe.Init (NomE: String);
{Construit un nouveau groupe}

 begin

  TElement.Init (Groupe, NomE);
  Effacable:= True;
  Lies:= New (PCollection, Init (15, 5));
  NbreAcces:= 0

 end;
```

```
Constructor TGroupe.Load (var flx: TStream);
{Lit un groupe dans un flux}

 var obj: PObject;

 begin

  TElement.Load (flx);
  flx.Read (Effacable, SizeOf (Effacable));
  obj:= flx.Get;
  Lies:= PCollection (obj);
  flx.Read (NbreActes, SizeOf (NbreActes));
  if NbreActes > 999999999 then NbreActes:= 0

 end;


Procedure TGroupe.Store (var flx: TStream);
{Ecrit un groupe dans un flux}

 begin

  TElement.Store (flx);
  flx.Write (Effacable, SizeOf (Effacable));
  flx.Put (Lies);
  flx.Write (NbreActes, SizeOf (NbreActes))

 end;


{*****************************************************************}


Constructor TStruct.Init;
{Initialise la structure}
 var flux: TBufStream;
     obj:  PObject;
     elem: PElement;
     grp:  PGroupe;

 Procedure Raccorder (g: PGroupe); far;
  Procedure Lier (r: PReference); far;
   var el: PElement;
   Function CorrespRef (e: PElement): Boolean; far;
    begin CorrespRef:= e^.Nom = r^.Nom end;
   begin el:= FirstThat (@CorrespRef); if el <> nil then r^.ref:=
el end;
  begin if g^.TypEl = Groupe then g^.lies^.ForEach (@Lier) end;

 begin

  TCollection.Init (30, 5);
  {grp:= New (PGroupe, Init ('Groupe Principal'));
   grp^.Effacable:= false; Insert (grp);}
  flux.Init ('Struct.dat', stOpenRead, 1024);
  if flux.Status = stOk then obj:= flux.Get;
  while flux.Status = stOk do begin
   elem:= PElement (obj);
   if elem^.typel = Groupe then begin grp:= PGroupe (elem); Insert
(grp) end
   else Insert (elem);
   obj:= flux.Get
  end;
  flux.Flush;
```

```
  ForEach (@Raccorder)

 end;


Destructor TStruct.Done;
{Détruit la structure}

 var structepuree: PCollection;
     flux:         TBufStream;

 Procedure SiGrpPack (elem: PElement); far;
  var liesepures: PCollection;
      grp: PGroupe;
  Procedure SiValidAj (refer: PReference); far;
   begin if refer^.ref^.Existence = true then liesepures^.Insert
(refer) end;
  begin
   if elem^.typel = Groupe then begin
    grp:= PGroupe (elem);
    liesepures:= New (PCollection, Init (10,5));
    grp^.lies^.ForEach (@SiValidAj);
    grp^.lies:= liesepures
   end
  end;
 Procedure SiExistAj (elem: PElement); far;
  var grp: PGroupe;
  begin
   if elem^.typel = Groupe then begin
    grp:= PGroupe (elem);
    if grp^.Existence = true then structepuree^.Insert (grp)
   end
   else if elem^.Existence = true then structepuree^.Insert (elem)
  end;
 Procedure EcrireElem (elem: PElement); far;
  var grp: PGroupe;
  begin
   if elem^.typel = Groupe then begin grp:= PGroupe (elem);
flux.Put (grp) end
   else flux.Put (elem)
  end;

 begin{TStruct.Done}

  ForEach (@SiGrpPack);
  structepuree:= New (PCollection, Init (30, 5));
  ForEach (@SiExistAj);
  flux.Init ('Struct.dat', stCreate, 1024);
  if flux.Status = stOk then begin
   structepuree^.ForEach (@EcrireElem);
   flux.Flush
  end;
  TCollection.Done

 end;


Function TStruct.ListeGrp: PCollection;
{Renvoie la liste des groupes de la structure}

 var coltmp: PCollection;

 Procedure SiGrpAj (e: PElement); far;
  begin
```

```
   If (e^.TypEl = Groupe) and (e^.Existence = True) then
coltmp^.Insert (e)
  end;

 begin

  coltmp:= New (PCollection, Init (15, 5));
  ForEach (@SiGrpAj);
  ListeGrp:= coltmp

 end;


Function TStruct.ListeGrpLies (Grp: String): PCollection;
{Renvoie la liste des groupes liés au groupe donné}

 var coltmp: PCollection;
     gp:     PGroupe;

 Function EstGrp (e: PElement): Boolean; far;
  begin EstGrp:= e^.Nom = Grp end;
 Procedure SiGrpAj (r: PReference); far;
  begin
   if (r^.Ref^.TypEl = Groupe) and (r^.Ref^.Existence = True) then
    coltmp^.Insert (r^.Ref)
  end;

 begin

  coltmp:= New (PCollection, Init (15, 5));
  gp:= FirstThat (@EstGrp);
  if gp <> nil then gp^.Lies^.ForEach (@SiGrpAj);
  ListeGrpLies:= coltmp

 end;


Function TStruct.ListeGrpNonLies (Grp: String): PCollection;
{Renvoie la liste des groupes non liés au groupe donné}

 var coltmp1, coltmp2, coltmp3: PCollection;

 Procedure SiPasDsLstAj (g: PGroupe); far;
  var gl: PGroupe;
  Function EstEl (g2: PGroupe): Boolean; far;
   begin EstEl:= g2^.Nom = g^.Nom end;
  begin
   gl:= coltmp2^.FirstThat (@EstEl);
   if gl=nil then if g^.Existence = True then coltmp3^.Insert (g)
  end;

 begin

  coltmp1:= ListeGrp;
  coltmp2:= ListeGrpLies (Grp);
  coltmp3:= New (PCollection, Init (15, 5));
  coltmp1^.ForEach (@SiPasDsLstAj);
  ListeGrpNonLies:= coltmp3

 end;


Function TStruct.ListeGrpEff: PCollection;
{Renvoie la liste des groupes effacés}
```

```
 var coltmp: PCollection;

 Procedure SiGEffAj (e: PElement); far;
  begin
   if (e^.TypEl = Groupe) and (e^.Existence = False) then
coltmp^.Insert (e)
  end;

 begin

  coltmp:= New (PCollection, Init (15, 5));
  ForEach (@SiGEffAj);
  ListeGrpEff:= coltmp

 end;


Function TStruct.ListePhr: PCollection;
{Renvoie la liste des phrases de la structure}

 var coltmp: PCollection;

 Procedure SiPhrAj (e: PElement); far;
  begin
   If (e^.TypEl = Phrase) and (e^.Existence = True) then
coltmp^.Insert (e)
  end;

 begin

  coltmp:= New (PCollection, Init (15, 5));
  ForEach (@SiPhrAj);
  ListePhr:= coltmp

 end;


Function TStruct.ListePhrLies (Grp: String): PCollection;
{Renvoie la liste des phrases liées au groupe donné}

 var coltmp: PCollection;
     gp:     PGroupe;

 Function EstGrp (e: PElement): Boolean; far;
  begin EstGrp:= e^.Nom = Grp end;
 Procedure SiPhrAj (r: PReference); far;
  begin
   if (r^.Ref^.TypEl = Phrase) and (r^.ref^.Existence = True) then
    coltmp^.Insert (r^.Ref)
  end;

 begin

  coltmp:= New (PCollection, Init (15, 5));
  gp:= FirstThat (@EstGrp);
  if gp <> nil then gp^.Lies^.ForEach (@SiPhrAj);
  ListePhrLies:= coltmp

 end;


Function TStruct.ListePhrNonLies (Grp: String): PCollection;
{Renvoie la liste des phrases non liées au groupe donné}
```

```
 var coltmp1, coltmp2, coltmp3: PCollection;

 Procedure SiPasDsLstAj (e: PElement); far;
  var el: PElement;
  Function EstEl (e2: PElement): Boolean; far;
   begin EstEl:= e2^.Nom = e^.Nom end;
  begin
   el:= coltmp2^.FirstThat (@EstEl);
   if el=nil then if e^.existence = True then coltmp3^.Insert (e)
  end;

 begin

  coltmp1:= ListePhr;
  coltmp2:= ListePhrLies (Grp);
  coltmp3:= New (PCollection, Init (15, 5));
  coltmp1^.ForEach (@SiPasDsLstAj);
  ListePhrNonLies:= coltmp3

 end;


Function TStruct.ListePhrEff: PCollection;
{Renvoie la liste des groupes effacés}

 var coltmp: PCollection;

 Procedure SiPEffAj (e: PElement); far;
  begin
   if (e^.TypEl = Phrase) and (e^.Existence = False) then
coltmp^.Insert (e)
  end;

 begin

  coltmp:= New (PCollection, Init (15, 5));
  ForEach (@SiPEffAj);
  ListePhrEff:= coltmp

 end;


Function TStruct.Principal: PGroupe;
{Renvoie le nom du groupe principal}

 var coltmp: PCollection;
     gp: PGroupe;

 Function EstPrinc (g: PGroupe): Boolean; far;
  begin EstPrinc:= g^.Effacable = False end;

 begin

  coltmp:= ListeGrp;
  gp:= coltmp^.FirstThat (@EstPrinc);
  Principal:= gp

 end;


Procedure TStruct.AjoutGrp (NomNouv: String; NomLie: String);
{Ajoute un groupe dans la structure}

 var g: PGroupe;
```

```
 function EstGrp (e: PElement): Boolean; far;
  begin EstGrp:= e^.Nom = NomNouv end;
 begin

  g:= FirstThat (@EstGrp);
  if g = nil then begin
   g:= New (PGroupe, Init (NomNouv));
   Insert (g);
  end;
  LierEl (NomNouv, NomLie)

 end;


Procedure TStruct.AjoutPhr (NomNouv: String; NomLie: String);
{Ajoute une phrase dans la structure}

 var p: PElement;

 function EstGrp (e: PElement): Boolean; far;
  begin EstGrp:= e^.Nom = NomNouv end;

 begin

  p:= FirstThat (@EstGrp);
  if p = nil then begin
   NomNouv:= NomNouv+'_';
   p:= New (PElement, Init (Phrase, NomNouv));
   Insert (p);
  end;
  LierEl (NomNouv, NomLie)

 end;


Procedure TStruct.SupprEl (NomSuppr: String);
{Supprime un élément de la structure}

 var el:  PElement;
     grp: PGroupe;

 Function EstEl (e: PElement): Boolean; far;
  begin EstEl:= e^.Nom = NomSuppr end;

 begin

  el:= FirstThat (@EstEl);
  if el <> nil then
   if el^.TypEl = Groupe then begin
    grp:= PGroupe (el);
    if grp^.Effacable = True then grp^.Existence:= False
   end
   else el^.Existence:= False

 end;


Procedure TStruct.RenEl (NouvNom: String; AncNom: String);
{Renomme un élément de la structure}

 var el, el2:  PElement;

 Function EstEl (e: PElement): Boolean; far;
  begin EstEl:= e^.Nom = AncNom end;
```

```
 Function EstEl2 (e: PElement): Boolean; far;
  begin EstEl2:= e^.Nom = NouvNom end;
 Procedure SiGrpEtLieChg (g: PGroupe); far;
  Procedure SiLienChg (r: PReference); far;
   begin if r^.ref^.Nom = NouvNom then r^.Nom:= NouvNom end;
  begin if g^.TypEl = Groupe then g^.Lies^.ForEach (@SiLienChg)
end;

 begin

  el:= FirstThat (@EstEl);
  el2:= FirstThat (@EstEl2);
  if (el <> nil) and (el2 = nil) then el^.Nom:= NouvNom;
  ForEach (@SiGrpEtLieChg)

 end;


Procedure TStruct.RecupEl (NomRecup: String);
{Recupère un élément supprimé de la structure}

 var el: PElement;

 Function EstEl (e: PElement): Boolean; far;
  begin EstEl:= e^.Nom = NomRecup end;

 begin

  el:= FirstThat (@EstEl);
  if el <> nil then el^.Existence:= True

 end;


Procedure TStruct.LierEl (el, gp: String);
{Lie l'élément donné au groupe donné}

 var g, g2:  PGroupe;
     e:  PElement;
     r:  PReference;

 Function EstGrp (el: PElement): Boolean; far;
  begin EstGrp:= el^.Nom = gp end;
 Function EstEl (e2: PElement): Boolean; far;
  begin EstEl:= e2^.Nom = el end;

 begin

  g:= FirstThat (@EstGrp);
  e:= FirstThat (@EstEl);
  if (g <> nil) and (e <> nil) then begin
   r:= New (PReference, Init (e));
   g^.lies^.Insert (r);
   if e^.TypEl = Groupe then begin
    r:= New (PReference, Init (g));
    g2:= PGroupe (e);
    g2^.lies^.Insert (r);
   end
  end

 end;


Procedure TStruct.DelierEl (el, gp: String);
```

```
{Délie l'élément donné du groupe donné}

 var g, g2:  PGroupe;
     e:  PElement;
     r:  PReference;

 Function EstGrp (e: PElement): Boolean; far;
  begin EstGrp:= e^.Nom = gp end;
 Function EstEl (e: PElement): Boolean; far;
  begin EstEl:= e^.Nom = el end;
 Function EstRefEl (re: PReference): Boolean; far;
  begin EstRefEl:= re^.Nom = el end;
 Function EstRefGp (re: PReference): Boolean; far;
  begin EstRefGp:= re^.Nom = gp end;

 begin

  g:= FirstThat (@EstGrp);
  if g <> nil then begin
   r:= g^.Lies^.FirstThat (@EstRefEl);
   if r <> nil then g^.Lies^.Free (r)
  end;
  e:= FirstThat (@EstEl);
  if e <> nil then if e^.TypEl = Groupe then begin
   g2:= PGroupe (e);
   r:= g2^.Lies^.FirstThat (@EstRefGp);
   if r <> nil then g2^.Lies^.Free (r)
  end

 end;


Procedure TStruct.Gest (AParent: PWindowsObject; TypGest: TypeEl);
{Ouvre la boîte de dialogue de gestion des groupes}

 var dlgtmp: PDlgGestGrp;

 begin

  dlgtmp:= New (PDlgGestGrp, Init (AParent, TypGest, @self));
  dlgtmp^.Execute

 end;


End.
```

Annexe 6 - La *unit GestPar*

```
{**************************}
{**  Fichier GestPar.pas  **}
{**************************}


Unit GestPar; {Gestion des paramètres utilisateurs}


Interface


Uses Objects, OWindows,
     MotVoc;


Type TInterf = (Simple, Picto, Groupes);

     PParam =  ^TParam;
     TParam =  Object (TObject){Les paramètres utilisateurs}

                typeinterf: TInterf;{Type d'interface}
                typevoix:   TVoix;{Type de voix}
                volume:     Integer;{Volume de prononciation}
                GrpG, GrpD: String;{Noms des groupes à gauche et à
droite}
                GrpGere:    String;{Nom du groupe en cours de ges-
tion}
                NumFich:    LongInt;{Nom du prochain fichier créé}
                NumPremF:   LongInt;{Nom du premier fichier de la
session}
                NumFichG:   LongInt;{Nom du prochain fichier de
test créé}

                Constructor Init;
                Destructor  Done; virtual;
                Constructor Load (var flx: TStream);
                Procedure   Store (var flx: TStream);
                Procedure Gest (AParent: PWindowsObject); virtual;

               end;


Const RParam: TStreamRec = (
       ObjType: 1005;
       VmtLink: Ofs (TypeOf (TParam)^);
       Load: @TParam.Load;
       Store: @TParam.Store);


Implementation


{$R GestPar.res}


Uses ODialogs, Strings,
     VarGlob, CyrTyp, GestGrp;
```

```
Const {Pour TDlgGestPar}
      id_rbGarcon = 118;
      id_rbFille =  119;
      id_rbHomme =  120;
      id_rbFemme =  121;
      id_rbPicto =   122;
      id_rbSimple =  123;
      id_rbGroupes = 124;
      id_stVolume = 125;
      id_sbVolume = 126;
      id_pbOk =      127;
      id_pbAnnuler = 128;
      id_pbAide =    129;


Type PDlgGestPar = ^TDlgGestPar;
     TDlgGestPar = Object (TDlgWindow)

                    Param: PParam;
                    rbGarcon, rbFille, rbHomme, rbFemme, rbPicto,
rbSimple, rbGroupes: PRadioButton;
                    stVolume: PCyrStatic;
                    sbVolume: PScrollBar;

                    Constructor Init (AParent: PWindowsObject;
Par: PParam);
                    Procedure   SetupWindow; virtual;
                    Procedure RegleVol (var Msg: TMessage);
                              virtual id_First+id_sbVolume;
                    Procedure Fermer (var Msg: TMessage);
                              virtual id_First+id_pbOk;
                    Procedure Annuler (var Msg: TMessage);
                              virtual id_First+id_pbAnnuler;
                    Procedure Aide (var Msg: TMessage);
                              virtual id_First+id_pbAide;

                   end;


{*************************************************************}
{TDlgGestPar}


Constructor TDlgGestPar.Init(AParent:PWindowsObject; Par: PParam);
{Construit la boîte de dialogue}

 begin

  TDlgWindow.Init (AParent, 'Parametres');
  Param:= Par;
  rbGarcon:= New (PRadioButton, InitResource (@self,
id_rbGarcon));
  rbFille:=  New (PRadioButton, InitResource (@self, id_rbFille));
  rbHomme:=  New (PRadioButton, InitResource (@self, id_rbHomme));
  rbFemme:=  New (PRadioButton, InitResource (@self, id_rbFemme));
  rbPicto:=  New (PRadioButton, InitResource (@self, id_rbPicto));
  rbSimple:=  New (PRadioButton, InitResource (@self,
id_rbSimple));
  rbGroupes:=  New (PRadioButton, InitResource (@self,
id_rbGroupes));
  stVolume:=New(PCyrStatic,InitResource (@self, id_stVolume, 11));
  sbVolume:= New (PScrollBar, InitResource (@self, id_sbVolume))

 end;
```

```
Procedure TDlgGestPar.SetupWindow;
{Met en place la boîte de dialogue}

 var strtmp: String;

 begin

  TDlgWindow.SetupWindow;
  Str (param^.volume, strtmp);
  strtmp:= 'Volume: '+strtmp;
  stVolume^.EcritStr (strtmp);
  sbVolume^.SetupWindow;
  sbVolume^.SetPosition (param^.volume);
  case Param^.TypeVoix of
   Garcon: rbGarcon^.SetCheck (bf_Checked);
   Fille: rbFille^.SetCheck (bf_Checked);
   Homme: rbHomme^.SetCheck (bf_Checked);
   Femme: rbFemme^.SetCheck (bf_Checked)
  end;
  case Param^.TypeInterf of
   Picto: rbPicto^.SetCheck (bf_Checked);
   Simple: rbSimple^.SetCheck (bf_Checked);
   Groupes: rbGroupes^.SetCheck (bf_Checked)
  end

 end;


Procedure TDlgGestPar.RegleVol (var Msg: TMessage);
{Gère la modification du volume}

 var strtmp: String;

 begin

  Str (sbVolume^.GetPosition, strtmp);
  strtmp:= 'Volume: '+strtmp;
  stVolume^.EcritStr (strtmp)

 end;


Procedure TDlgGestPar.Fermer (var Msg: TMessage);
{Ferme le dialogue après avoir récupéré les modifications}

 begin

  param^.Volume:= sbVolume^.GetPosition;
  if rbGarcon^.GetCheck = bf_Checked then param^.TypeVoix:= Garcon
  else if rbFille^.GetCheck = bf_Checked then param^.TypeVoix:=
Fille
  else if rbHomme^.GetCheck = bf_Checked then param^.TypeVoix:=
Homme
  else if rbFemme^.GetCheck = bf_Checked then param^.TypeVoix:=
Femme;
  if rbPicto^.GetCheck = bf_Checked then param^.TypeInterf:= Picto
  else if rbSimple^.GetCheck = bf_Checked then param^.TypeInterf:=
Simple
  else if rbGroupes^.GetCheck = bf_Checked then pa-
ram^.TypeInterf:= Groupes;
  if canclose then EndDlg (id_pbOk)

 end;
```

```
Procedure TDlgGestPar.Annuler (var Msg: TMessage);
{Ferme le dialogue sans enregistrer les modifications}

 begin if canclose then EndDlg (id_pbAnnuler) end;


{*************************************************************}
{TParam}


Constructor TParam.Init;
{Initialise les paramètres}

 var flux:   TBufStream;
     obj:    PObject;
     g1, g2: PGroupe;

 Function EstGrpG (e: PElement): Boolean; far;
  begin EstGrpG:= e^.Nom = GrpG end;
 Function EstGrpD (e: PElement): Boolean; far;
  begin EstGrpD:= e^.Nom = GrpD end;

 begin

  TObject.Init;
  {typeinterf:= Simple; typevoix:= Homme; volume:= 50;
   GrpG:='Groupe Principal';GrpD:='';GrpGere:='Groupe Principal';
   NumFich:= 0; NumPremF:= 0; NumFichG:= 0}
  flux.Init ('Param.dat', stOpenRead, 1024);
  if flux.Status = stOk then obj:= flux.Get;
  if flux.Status = stOk then self:= PParam (obj)^;
  flux.Done;
  g1:= StructGrp^.FirstThat (@EstGrpG);
  g2:= StructGrp^.FirstThat (@EstGrpD);
  if (g1 = nil) and (g2 = nil) then GrpG:= 'Groupe Principal';
  if NumFich > 9999999 then NumFich:= 0;
  if NumFichG > 999999 then NumFichG:= 0;
  NumPremF:= NumFich

 end;


Destructor TParam.Done;
{Sauve puis détruit les paramètres}

 var flux: TBufStream;

 begin

  flux.Init ('Param.dat', stCreate, 1024);
  if flux.Status = stOk then flux.Put (@self);
  flux.Done;
  TObject.Done

 end;


Constructor TParam.Load (var flx: TStream);
{Lit les paramètres dans un flux}

 var car:    Char;
     i:      Integer;
     pchtmp: Array [0..255] of Char;
```

```
 begin

  flx.Read (car, SizeOf (car));
  case car of
   'S': typeinterf:= Simple;
   'P': typeinterf:= Picto;
   'G': typeinterf:= Groupes
  end;
  flx.Read (car, SizeOf (car));
  case car of
   'H': typevoix:= Homme;
   'F': typevoix:= Femme;
   'G': typevoix:= Garcon;
   'I': typevoix:= Fille
  end;
  flx.Read (volume, SizeOf (volume));
  for i:= 0 to 255 do pchtmp [i]:= ' ';
  flx.Read (pchtmp, SizeOf (pchtmp));
  GrpG:= StrPas (pchtmp);
  for i:= 0 to 255 do pchtmp [i]:= ' ';
  flx.Read (pchtmp, SizeOf (pchtmp));
  GrpD:= StrPas (pchtmp);
  for i:= 0 to 255 do pchtmp [i]:= ' ';
  flx.Read (pchtmp, SizeOf (pchtmp));
  GrpGere:= StrPas (pchtmp);
  flx.Read (NumFich, SizeOf (NumFich));
  flx.Read (NumFichG, SizeOf (NumFichG));

 end;


Procedure TParam.Store (var flx: TStream);
{Ecrit les paramètres dans un flux}

 var car: Char;
     i:   Integer;
     pchtmp: Array [0..255] of Char;

 begin

  case typeinterf of
   Simple: car:= 'S';
   Picto:  car:= 'P';
   Groupes: car:= 'G'
  end;
  flx.Write (car, SizeOf (car));
  case typevoix of
   Homme:  car:= 'H';
   Femme:  car:= 'F';
   Garcon: car:= 'G';
   Fille:  car:= 'I'
  end;
  flx.Write (car, SizeOf (car));
  flx.Write (volume, SizeOf (volume));
  for i:= 0 to 255 do pchtmp [i]:= ' ';
  StrPCopy (pchtmp, GrpG);
  flx.Write (pchtmp, SizeOf (pchtmp));
  for i:= 0 to 255 do pchtmp [i]:= ' ';
  StrPCopy (pchtmp, GrpD);
  flx.Write (pchtmp, SizeOf (pchtmp));
  for i:= 0 to 255 do pchtmp [i]:= ' ';
  StrPCopy (pchtmp, GrpGere);
  flx.Write (pchtmp, SizeOf (pchtmp));
  flx.Write (NumFich, SizeOf (NumFich));
```

```
  flx.Write (NumFichG, SizeOf (NumFichG));

 end;


Procedure TParam.Gest (AParent: PWindowsObject);
{Ouvre la boîte de dialogue de gestion des paramètres}

 var dlgtmp: PDlgGestPar;

 begin

  dlgtmp:= New (PDlgGestPar, Init (AParent, @self));
  dlgtmp^.Execute

 end;


End.
```

Annexe 7 - Les *units* annexes

```
{**************************}
{**  Fichier VarGlob.pas  **}
{**************************}


Unit VarGlob;{Variables globales du programme}


Interface


Uses WinTypes,
     GestPar, GestGrp;


Const NomAppl: PChar = 'Cyrano';{Nom de l'application}


Var Ecran: HWnd;{Adresse de l'image de l'écran derrière notre ap-
plication}

    CurseurFleche: HCursor;{Flèche de pointage}
    CurseurHP:     HCursor;{Haut Parleur de prononciation}

    Param: PParam;{Pointeur vers les paramètres de fonctionnement}
    StructGrp: PStruct;{Pointeur vers la structure de groupes}


{*************************************************************}


Implementation


End.
```

```
{*************************}
{**  Fichier CyrTyp.pas  **}
{*************************}


Unit CyrTyp;{Types propres à Cyrano}


Interface


Uses ODialogs, OWindows,
     WinTypes;


Type PCyrStatic = ^TCyrStatic;
     TCyrStatic = Object (TStatic)

                    Procedure EcritStr (Texte: String); virtual;

                   end;


     PCyrEdit = ^TCyrEdit;
     TCyrEdit = Object (TEdit)

                 stCible: PCyrStatic;
                 Texte: String;

                 Constructor InitResource (AParent: PWindowsOb-
ject; ResourceID, ATextLen: Word; St: PCyrStatic; Txt: String);
                 Procedure wmMouseMove (var Msg: TMessage);
                           virtual wm_First+wm_MouseMove;
                 Procedure LitStr (var Txt: String); virtual;
                 Procedure EcritStr (Txt: String); virtual;
                 Procedure AjouteStr (Txt: String); virtual;

                end;


     PCyrButton = ^TCyrButton;
     TCyrButton = Object (TButton)

                    stCible: PCyrStatic;
                    Texte: String;

                    Constructor InitResource (AParent: PWindowsOb-
ject; ResourceID: Word; St: PCyrStatic; Txt: String);
                    Procedure wmMouseMove (var Msg: TMessage);
                              virtual wm_First+wm_MouseMove;

                   end;


     PCyrListBox = ^TCyrListBox;
     TCyrListBox = Object (TListBox)

                     stCible: PCyrStatic;
                     Texte: String;

                     Constructor InitResource (AParent: PWindowsOb-
ject; ResourceID: Word; St: PCyrStatic; Txt: String);
                     Procedure wmMouseMove (var Msg: TMessage);
                               virtual wm_First+wm_MouseMove;
```

```
                    Procedure EcritStr (Txt: String; Ind: Inte-
ger); virtual;
                    Procedure LitStr (var Txt: String; Ind: Inte-
ger); virtual;

                  end;


     PCyrScrollBar = ^TCyrScrollBar;
     TCyrScrollBar = Object (TScrollBar)

                      stCible: PCyrStatic;
                      Texte: String;

                      Constructor InitResource (AParent: PWin-
dowsObject; ResourceID: Word; St: PCyrStatic; Txt: String);
                      Procedure wmMouseMove (var Msg: TMessage);
                                virtual wm_First+wm_MouseMove;

                     end;


     PCyrComboBox = ^TCyrComboBox;
     TCyrComboBox = Object (TComboBox)

                      stCible: PCyrStatic;
                      Texte: String;

                      constructor InitResource(AParent: PWindowsOb-
ject; ResourceID: Integer; ATextLen: Word; St: PCyrStatic; Txt:
String);
                      Procedure wmMouseMove (var Msg: TMessage);
                                virtual wm_First+wm_MouseMove;

                    Procedure EcritStr (Txt: String; Ind: Inte-
ger); virtual;
                    Procedure LitStr (var Txt: String; Ind: Inte-
ger); virtual;
                    Procedure SelectStr (Txt: String); virtual;

                    end;


{*****************************************************************}


Implementation


Uses Strings;


{TCyrStatic}


Procedure TCyrStatic.EcritStr (Texte: String);

 var i:        Integer;
     pchtmp1:  Array [0..255] of Char;
     pchtmp2:  Array [0..255] of Char;

 begin
```

```
  for i:= 0 to 255 do pchtmp1[i]:= ' ';
  StrPCopy (pchtmp1, Texte);
  for i:= 0 to 255 do pchtmp2[i]:= ' ';
  GetText (pchtmp2, 255);
  if StrComp (pchtmp1, pchtmp2) <> 0 then SetText (pchtmp1)

 end;


{*************************************************************}
{TCyrEdit}


Constructor TCyrEdit.InitResource (AParent: PWindowsObject; Re-
sourceID, ATextLen: Word; St: PCyrStatic; Txt: String);

 begin

  TEdit.InitResource (AParent, ResourceID, ATextLen);
  stCible:= St;
  Texte:= Txt

 end;


Procedure TCyrEdit.wmMouseMove (var Msg: TMessage);

 begin

  DefWndProc (Msg);
  if stCible <> nil then stCible^.EcritStr (Texte)

 end;


Procedure TCyrEdit.LitStr (var Txt: String);

 var i:      Integer;
     pchtmp: Array [0..255] of Char;

 begin

  for i:= 0 to 255 do pchtmp[i]:= ' ';
  GetText (pchtmp, 255);
  Txt:= StrPas (pchtmp)

 end;


Procedure TCyrEdit.EcritStr (Txt: String);

 var i:      Integer;
     pchtmp: Array [0..255] of Char;

 begin

  for i:= 0 to 255 do pchtmp[i]:= ' ';
  StrPCopy (pchtmp, Txt);
  SetText (pchtmp)

 end;


Procedure TCyrEdit.AjouteStr (Txt: String);
```

```
 var i:      Integer;
     pchtmp: Array [0..255] of Char;

 begin

  for i:= 0 to 255 do pchtmp[i]:= ' ';
  StrPCopy (pchtmp, Txt);
  Insert (pchtmp)

 end;


{*****************************************************************}
{TCyrButton}


Constructor TCyrButton.InitResource (AParent: PWindowsObject;
            ResourceID: Word; St: PCyrStatic; Txt: String);

 begin

  TButton.InitResource (AParent, ResourceID);
  stCible:= St;
  Texte:= Txt

 end;


Procedure TCyrButton.wmMouseMove (var Msg: TMessage);

 begin

  DefWndProc (Msg);
  if stCible <> nil then stCible^.EcritStr (Texte)

 end;


{*****************************************************************}
{TCyrListBox}


Constructor TCyrListBox.InitResource (AParent: PWindowsObject;
            ResourceID: Word; St: PCyrStatic; Txt: String);

 begin

  TListBox.InitResource (AParent, ResourceID);
  stCible:= St;
  Texte:= Txt

 end;


Procedure TCyrListBox.wmMouseMove (var Msg: TMessage);

 begin

  DefWndProc (Msg);
  if stCible <> nil then stCible^.EcritStr (Texte)

 end;
```

```
Procedure TCyrListBox.EcritStr (Txt: String; Ind: Integer);

 var i:      Integer;
     pchtmp: Array [0..255] of Char;

 begin

  for i:= 0 to 255 do pchtmp[i]:= ' ';
  StrPCopy (pchtmp, Txt);
  if Ind = -1 then AddString (pchtmp)
  else InsertString (pchtmp, Ind)

 end;


Procedure TCyrListBox.LitStr (var Txt: String; Ind: Integer);

 var i:      Integer;
     pchtmp: Array [0..255] of Char;

 begin

  for i:= 0 to 255 do pchtmp[i]:= ' ';
  GetString (pchtmp, Ind);
  Txt:= StrPas (pchtmp)

 end;


{*****************************************************************}
{TCyrScrollBar}


Constructor TCyrScrollBar.InitResource (AParent: PWindowsObject;
            ResourceID: Word; St: PCyrStatic; Txt: String);

 begin

  TScrollBar.InitResource (AParent, ResourceID);
  stCible:= St;
  Texte:= Txt

 end;


Procedure TCyrScrollBar.wmMouseMove (var Msg: TMessage);

 begin

  DefWndProc (Msg);
  if stCible <> nil then stCible^.EcritStr (Texte)

 end;


{*****************************************************************}
{TCyrComboBox}


Constructor TCyrComboBox.InitResource(AParent: PWindowsObject;
ResourceID: Integer; ATextLen: Word; St: PCyrStatic; Txt: String);

 begin
```

```
  TComboBox.InitResource (AParent, ResourceID, ATextLen);
  stCible:= St;
  Texte:= Txt

 end;


Procedure TCyrComboBox.wmMouseMove (var Msg: TMessage);

 begin

  DefWndProc (Msg);
  if stCible <> nil then stCible^.EcritStr (Texte)

 end;


Procedure TCyrComboBox.EcritStr (Txt: String; Ind: Integer);

 var i:      Integer;
     pchtmp: Array [0..255] of Char;

 begin

  for i:= 0 to 255 do pchtmp[i]:= ' ';
  StrPCopy (pchtmp, Txt);
  if Ind = -1 then AddString (pchtmp)
  else InsertString (pchtmp, Ind)

 end;


Procedure TCyrComboBox.LitStr (var Txt: String; Ind: Integer);

 var i:      Integer;
     pchtmp: Array [0..255] of Char;

 begin

  for i:= 0 to 255 do pchtmp[i]:= ' ';
  GetString (pchtmp, Ind);
  Txt:= StrPas (pchtmp)

 end;


Procedure TCyrComboBox.SelectStr (Txt: String);

 var i:      Integer;
     pchtmp: Array [0..255] of Char;

 begin

  for i:= 0 to 255 do pchtmp[i]:= ' ';
  StrPCopy (pchtmp, Txt);
  SetSelString (pchtmp, -1)

 end;


End.
```